VMS Version 5.1
New Features Manual

Order Number: AA–MG29A–TE

December 1988

This manual describes new features of VMS and DECwindows for VMS Version 5.1.

Revision/Update Information: This is a new manual.

Software Version: VMS Version 5.1

digital equipment corporation
maynard, massachusetts

December 1988

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

ZK5000

Production Note

This book was produced with the VAX DOCUMENT electronic publishing system, a software tool developed and sold by DIGITAL. In this system, writers use an ASCII text editor to create source files containing text and English-like code; this code labels the structural elements of the document, such as chapters, paragraphs, and tables. The VAX DOCUMENT software, which runs on the VMS operating system, interprets the code to format the text, generate a table of contents and index, and paginate the entire document. Writers can print the document on the terminal or line printer, or they can use DIGITAL-supported devices, such as the LN03 laser printer and PostScript printers (PrintServer 40 or LN03R ScriptPrinter), to produce a typeset-quality copy containing integrated graphics.

Contents

FIGURES

TABLES

Preface

This manual includes functional VMS Version 5.1 enhancements that support the new DECwindows environment. Components with new features include VAXTPU, RMS, CDA, and DCL. The following types of enhancements have been made:

- Enhancements that provide support for DECwindows graphic and windowing capabilities.
- Enhancements to support a new VMS file format. The new file format allows encoding of compound documents. Compound documents are files consisting of a number of integrated components such as text, graphics, and scanned images.

The following four DCL commands and one DCL qualifier have been added:

- CONVERT/DOCUMENT command
- EXCHANGE/NETWORK command
- SET DISPLAY command
- SHOW DISPLAY command
- SET FILE/SEMANTICS qualifier

Intended Audience

This manual is intended for all system users of VMS and DECwindows. However, the VAXTPU section of this manual is for experienced VAXTPU programmers.

Document Structure

This manual consists of the following chapters:

- Chapters 1–7 describe the VAXTPU and EVE new features.
- Chapter 8 describes the CDA new features.
- Chapter 9 describes the VMS RMS new features.
- Chapter 10 describes the new DCL command EXCHANGE/NETWORK.
- Chapter 11 describes the new DCL commands SET DISPLAY and SHOW DISPLAY.

Associated Documents

For additional information about topics discussed in this document, see the following manuals:

- *VMS Compound Document Architecture Manual*
- *VMS Version 5.1 Release Notes*
- *VMS DECwindows User's Guide*
- *VMS DECwindows Guide to Application Programming*
- *XUI Style Guide*
- *VAX Text Processing Utility Manual*

Conventions

The following conventions are used in this manual:

mouse	The term *mouse* is used to refer to any pointing device, such as a mouse, a puck, or a stylus.
MB1, MB2, MB3	MB1 indicates the left mouse button, MB2 indicates the middle mouse button, and MB3 indicates the right mouse button. (The buttons can be redefined by the user.)
PB1, PB2, PB3, PB4	PB1, PB2, PB3, and PB4 indicate buttons on the puck.
SB1, SB2	SB1 and SB2 indicate buttons on the stylus.
Ctrl/x	A sequence such as Ctrl/x indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 x	A sequence such as PF1 x indicates that you must first press and release the key labeled PF1, then press and release another key or a pointing device button.
Return	A key name is shown enclosed to indicate that you press a key on the keyboard.
. . .	In examples, a horizontal ellipsis indicates one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
()	In format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.
[]	In format descriptions, brackets indicate that whatever is enclosed is optional; you can select none, one, or all of the choices. Brackets enclosing a comma and horizontal ellipsis mean that you can repeat the preceding item one or more times, separating two or more items with commas.

[[, . . .]]	Double brackets in examples show an optional portion of the format of a built-in procedure or lexical element. When double brackets enclose an item or series of items, you can select one of the items. Double brackets enclosing a comma and horizontal ellipsis mean that you can repeat the preceding item one or more times, separating two or more items with commas. For example: `parameter [[,...]]`
{}	In format descriptions, braces surround a required choice of options; you must choose one of the options listed. Braces enclose a mandatory portion of the format of a built-in procedure or lexical element. When braces enclose a stacked list of items, you must choose one of the items. For example: { string / range }
red ink	Red ink indicates information that you must enter from the keyboard or a screen object that you must choose or click on. For online versions, user input is shown in **bold**.
boldface text	Boldface text represents the introduction of a new term.
italic text	Italic text represents information that can vary in system messages (for example, Internal error *number*).
UPPERCASE TEXT	Uppercase letters indicate that you must enter a command (for example, enter OPEN/READ). Uppercase letters can also indicate the name of a routine, the name of a file, the name of a file protection code, or the abbreviation for a system privilege.

UPPERCASE letters and special symbols	Uppercase letters and special symbols in syntax descriptions and sample procedures indicate VAXTPU reserved words and predeclared identifiers, and other user input which must be typed exactly as shown. For example:
	
	`PROCEDURE`
	`UNDERLINE`
	
	String constants are shown in lowercase to emphasize that they are strings. However, they, too, must be typed exactly as shown.

lowercase letters	Lowercase letters in syntax descriptions and sample procedures represent elements you must replace according to the description in the text. For example, when a data type, such as buffer, is used in a syntax example, replace it with the variable name assigned to the data item when it was created. In the following assignment statement, *my_buffer_variable* is the variable name assigned to the buffer you are creating:
	
	`my_buffer_variable :=`
	`CREATE_BUFFER ('my_buf_name', 'my_file_name')`
	
	To specify a buffer as a parameter for a VAXTPU built-in procedure, use the variable for the buffer. For example, to erase the contents of the buffer created in the preceding statement, enter the following:
	
	`ERASE (my_buffer_variable)`

user_	The prefix *user_* is used in many sample procedures as a part of the procedure name. DIGITAL suggests that you replace the prefix *user* with your initials. This (or another convention of your choice) helps to ensure that the variables and procedure names that you create do not conflict with either VAXTPU built-in procedure names, or the procedure names and variables of your editing interface.

1 Overview of the DECwindows VAX Text Processing Utility

This chapter presents an overview of the VAX Text Processing Utility (VAXTPU) Version 2.2, released with VMS Version 5.1.

This chapter discusses the following:

- The relationship between this version of VAXTPU and other versions of VAXTPU
- The DECwindows version of VAXTPU
- The relationship between DECwindows VAXTPU and other DECwindows features (such as the XUI Toolkit or Xlib)
- The relationship between DECwindows VAXTPU and the DECwindows User Interface Language (UIL)

Note: The VAXTPU section of this manual is intended for experienced VAXTPU programmers who have read the *VMS DECwindows Guide to Application Programming* and who have access to the *VAX Text Processing Utility Manual.*

1.1 Relationship to Previous Versions of VAXTPU

This section of the *VMS Version 5.1 New Features Manual* documents VAXTPU Version 2.2, which is the version of VAXTPU released with VMS Version 5.1. If you need to submit a Software Problem Report (SPR) on the version of VAXTPU released with VMS Version 5.1, please state in the SPR that you are reporting a problem with VAXTPU Version 2.2.

VAXTPU Version 2.2 includes a DECwindows version of VAXTPU and a non-DECwindows version of VAXTPU. You can use the DECwindows version of VAXTPU only on a workstation that supports DECwindows. For information about how to invoke the DECwindows version of VAXTPU, see Chapter 3.

Both versions of VAXTPU released with VMS Version 5.1 include all the built-in procedures and features provided in the version of VAXTPU released with VMS Version 5.0, plus some new built-in procedures and other features. This manual documents only the features of VAXTPU that are new in the version of VAXTPU released with VMS 5.1. Use this manual as a supplement to the *VAX Text Processing Utility Manual.*

Some new VAXTPU features are available in both the DECwindows and the non-DECwindows versions of VAXTPU. Other new VAXTPU features are available only in the DECwindows version of VAXTPU. All new features of VAXTPU are available in the DECwindows version.

The difference between the DECwindows version of VAXTPU 2.2 and the non-DECwindows version of VAXTPU 2.2 is that DECwindows VAXTPU includes the built-in procedures needed to implement a DECwindows text processing interface, while the non-DECwindows version does not include these built-ins.

Note that the "windows" referred to in the product name *DECwindows* are not the same as VAXTPU windows, which have been supported in VAXTPU for several releases. For more information about the difference between DECwindows windows and VAXTPU windows, see Chapter 6.

Any VAXTPU program or package written using the version of VAXTPU released with VMS Version 5.0 will still work under either version of VAXTPU released with VMS Version 5.1, even if the old program or package does not implement a DECwindows text processing interface.

1.2 The DECwindows Version of VAXTPU

The DECwindows version of the VAX Text Processing Utility (VAXTPU) is a programmable utility that includes the following tools:

- A language (a set of statements and reserved words plus a set of syntax rules)
- An interpreter
- A compiler
- A callable interface
- A default editing interface, the Extensible VAX Editor (EVE), written in the VAXTPU language
- A set of built-in procedures that you can use to implement a DECwindows text processing interface

DECwindows VAXTPU is designed as a tool to aid application and system programmers in developing text editors and other text-manipulating products. Using programs composed of VAXTPU statements, you can create products that have DECwindows interfaces. For more information about the DECwindows interfaces, see the *XUI Style Guide*.

You can also use DECwindows VAXTPU to create text processing products with command-line interfaces (interfaces that you control by typing in commands).

For general information about creating editors and other products using VAXTPU, see the *VAX Text Processing Utility Manual*.

1.3 DECwindows VAXTPU and DECwindows Features

The DECwindows environment has a number of toolkits and libraries containing routines for creating and manipulating DECwindows interfaces. For example, DECwindows routines allow you to create and manipulate clipboard entries, global selections, and widgets. For an overview of the DECwindows libraries and toolkits, see *VMS DECwindows Guide to Application Programming*.

1.3 DECwindows VAXTPU and DECwindows Features

DECwindows VAXTPU contains a number of built-in procedures that call some of the routines in the various DECwindows libraries and toolkits.

Using these DECwindows VAXTPU built-in procedures, you can create and manipulate various features of a DECwindows interface from within a VAXTPU program. For a list of the kinds of widgets you can create and manipulate using VAXTPU built-in procedures, see Chapter 7. In most cases, you use VAXTPU DECwindows built-in procedures without needing to know what DECwindows routine a given built-in procedure calls.

You cannot directly call DECwindows routines (such as XUI Toolkit or Xlib Toolkit routines) from within a program written in the VAXTPU language. To use a DECwindows routine in a VAXTPU program, you can use one or more of the following techniques:

- Use a VAXTPU built-in procedure that calls a DECwindows routine. Examples of such VAXTPU built-in procedures include the following:
  - CREATE_WIDGET
  - DELETE (WIDGET)
  - MANAGE_WIDGET
  - SET (DRM_HIERARCHY)
  - SET (WIDGET)
  - SET (WIDGET_CALLBACK)
  - UNMANAGE_WIDGET

  For more information about how to use the DECwindows built-ins in VAXTPU, see the individual built-in descriptions in Chapter 5. For more information about the types of widget resource values supported by VAXTPU, see Chapter 7.
- Using a compiled language that follows the VMS calling standard, write a function calling the desired XUI Toolkit routine. You can then use the built-in procedure CALL_USER in your VAXTPU program to invoke the program written in the non-VAXTPU language. For more information about using the built-in procedure CALL_USER, see the *VAX Text Processing Utility Manual.*
- Using a compiled language that follows the VMS calling standard, write a program calling the desired XUI Toolkit routine. You can then use the VAXTPU callable interface to invoke the program written in the non-VAXTPU language. For more information about using the VAXTPU callable interface, see the *VMS Utility Routines Manual.*

The DECwindows version of VAXTPU 2.2 does not supply built-in procedures implementing all DECwindows routines. For example, there are no VAXTPU built-in procedures to handle pixmaps or floating-point numbers or to manipulate entities such as lines, curves, and fonts.

However, the DECwindows version of VAXTPU 2.2 allows you to create a wide variety of widgets, to designate callback routines for those widgets, to fetch and set geometry- and text-related resources of the widgets, and to perform other functions related to creating a DECwindows text processing

interface. For example, the DECwindows EVE editor is a text processing interface created with DECwindows VAXTPU.

1.4 DECwindows VAXTPU and the DECwindows User Interface Language

You can use VAXTPU programs with DECwindows User Interface Language (UIL) files just as you would use programs in any other language with UIL files. For an example of a VAXTPU program and a UIL file designed to be used together, see the description of the CREATE_WIDGET built-in in Chapter 5. For more information about using UIL files in conjunction with programs written in other languages, see *VMS DECwindows Guide to Application Programming*.

2 Invoking DECwindows VAXTPU with FileView

In the DECwindows version of VAXTPU, you can invoke VAXTPU by using FileView, a menu-driven interface for choosing and using files. DIGITAL recommends that you specify only one file at a time when you invoke VAXTPU this way. For general information about using FileView, see the *VMS DECwindows User's Guide*.

To invoke DECwindows VAXTPU using FileView, take the following steps:

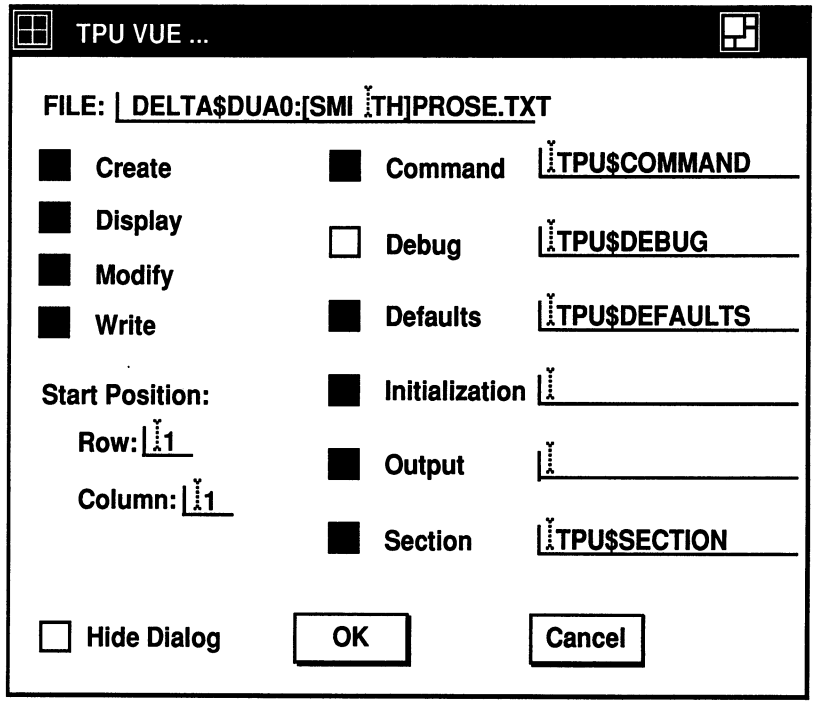
1. Choose the Files menu from the FileView menu bar.
2. From the Files menu, choose the Edit menu item.

   FileView displays the Edit dialog box with a field in which to specify the file to edit and a set of radio buttons with which to choose the editor.
3. Optionally, supply the specification of a file to edit if FileView has not supplied one.
4. Choose the EVE editor.
5. Click on OK.

   FileView displays a dialog box allowing you to invoke VAXTPU with any command qualifiers except the /[NO]READ, /[NO]RECOVER, and /[NO]JOURNAL qualifiers. The /[NO]JOURNAL and /[NO]RECOVER qualifiers do not appear in the dialog box because the DECwindows version of VAXTPU does not support journaling. The /[NO]READ qualifier does not appear because /[NO]READ can interact with /[NO]WRITE in unproductive ways if specified incorrectly. You can accomplish any result you need using /[NO]WRITE. For more information about the use of these qualifiers, see the *VAX Text Processing Utility Manual*.

Figure 2–1 shows the layout of the dialog box used to invoke VAXTPU.

Figure 2–1 Layout of the VAXTPU Dialog Box



ZK–0151A–GE

3 Changes to Command Line Qualifiers and Data Types

This chapter provides information on the following subjects:

- Behavior of the /DISPLAY command qualifier in VAXTPU Version 2.2
- New and modified data types in VAXTPU Version 2.2

3.1 Behavior of the /DISPLAY Command Qualifier

To choose the DECwindows or the non-DECwindows version of VAXTPU, use the command qualifier /DISPLAY on the DCL command line when you invoke VAXTPU.

VAXTPU supplies two keywords that you can use for specifying the version you want. The syntax for the /DISPLAY qualifier is as follows:

```
[ /DISPLAY [ = CHARACTER_CELL ] ]
[ /NODISPLAY [ = DECWINDOWS   ] ]
```

The /DISPLAY command qualifier is optional. By default, VAXTPU uses the non-DECwindows version, regardless of whether you are running VAXTPU on a workstation or a terminal.

If you specify /DISPLAY = CHARACTER_CELL, VAXTPU uses its character-cell screen manager, which implements the non-DECwindows version of VAXTPU by running in a DECterm (or VWS) terminal emulator or on a physical terminal.

If you specify /DISPLAY = DECWINDOWS, and if the DECwindows environment is available, VAXTPU uses the DECwindows screen manager, which creates a DECwindows window in which to run VAXTPU.

If you specify /DISPLAY = DECWINDOWS and the DECwindows environment is not available, VAXTPU uses its character-cell screen manager to implement the non-DECwindows version of VAXTPU.

To override the default without using the /DISPLAY qualifier, define the logical name TPU$DISPLAY_MANAGER to be DECWINDOWS, as follows:

```
$ DEFINE TPU$DISPLAY_MANAGER DECWINDOWS
```

For more information about the difference between a DECwindows window and a VAXTPU window, see Chapter 6.

For more information about the /[NO]DISPLAY qualifier, see the *VAX Text Processing Utility Manual*.

3.2 New and Modified Data Types

This section covers data types that are new or modified in VAXTPU Version 2.2. For more information about VAXTPU data types, see the *VAX Text Processing Utility Manual.*

3.2.1 Widgets

The DECwindows version of VAXTPU Version 2.2 provides the widget data type to support DECwindows widgets. The non-DECwindows version of VAXTPU Version 2.2 does not support this data type.

A widget is a structure used as an interaction mechanism by which users give input to an application or receive messages from an application. For more information about what a widget is, see the *VMS DECwindows Guide to Application Programming.*

You can use the equal operator (=) or the not-equal operator (<>) on widgets to determine whether they are equal (that is, whether they are the same widget instance), but you cannot use any other relational or arithmetic operators on them. For more information about the difference between a class of widgets and a widget instance, see the *VMS DECwindows Guide to Application Programming.*

Once you have created a widget instance, VAXTPU does not delete the widget instance, even if there are no variables referencing it. To delete a widget, use the DELETE (widget_variable) built-in.

The DECwindows version of VAXTPU provides the same support for DECwindows gadgets that it provides for widgets. A gadget is a structure similar to a widget, but is not associated with its own unique DECwindows window. Gadgets do not require as much memory to implement as widgets do. In most cases, you can use the same DECwindows VAXTPU built-ins on gadgets that you use on widgets. For more information about gadgets, see the *VMS DECwindows Guide to Application Programming.*

3.2.2 String Data Types

Both the DECwindows and non-DECwindows versions of VAXTPU Version 2.2 allow you to replicate and reduce character strings.

To replicate a string, specify the string to be reproduced, then the multiplication operator (*), and then the number of times you want the string to be replicated. For example, the following VAXTPU statement inserts 10 underscores into the current buffer at the editing point:

```
COPY_TEXT ("_" * 10)
```

Note that the string to be replicated must be on the left-hand side of the operator. For example, the following VAXTPU statement produces an error:

```
COPY_TEXT (10 * "_")
```

To reduce a string, specify the string to be modified, then the subtraction operator (-), and then the substring to be removed. For example, the following table shows the effects of two string-reduction operations:

VAXTPU Statement	Result
COPY_TEXT ("FILENAME.MEM" - "FILE")	Inserts the string "NAME.MEM" into the current buffer at the editing point.
COPY_TEXT ("LISTINGS.LIS" - "LIS")	Inserts the string "TINGS.LIS" into the current buffer at the editing point.

4 New and Modified Built-In Procedures Common to Both the DECwindows and Non-DECwindows Versions of VAXTPU

This chapter describes the new and modified built-in procedures that are common to both the DECwindows and non-DECwindows versions of VAXTPU Version 2.2. For information about the new DECwindows-related built-ins in VAXTPU Version 2.2, see Chapter 5. For information about the VAXTPU built-ins released before VMS Version 5.1, see the *VAX Text Processing Utility Manual*. For an overview of recent VAXTPU release history and version numbering, see Chapter 1.

4.1 Descriptions of VAXTPU Built-In Procedures Common to DECwindows and Non-DECwindows

Using the new and modified built-ins described in this section, you can perform the following tasks:

- Convert location specifications from one coordinate system to another
- Dynamically alter a range
- Fetch information on window boundaries and size
- Move the editing point to a specified record
- Control whether a buffer is marked as modified

For more information about using VAXTPU built-ins, see the *VAX Text Processing Utility Manual*.

CONVERT

Given the coordinates of a point in one coordinate system, returns the corresponding coordinates for the point in the coordinate system you specify.

FORMAT

CONVERT ({ *DECW_ROOT_WINDOW* | *SCREEN* | *window* }, { *CHARACTERS,* | *COORDINATES,* } *from_x_integer, from_y_integer,* { *DECW_ROOT_WINDOW* | *SCREEN* | *window* }, { *CHARACTERS,* | *COORDINATES,* } *to_x_integer, to_y_integer*)

PARAMETERS

DECW_ROOT_WINDOW

Specifies the coordinate system to be that used by the root window of the screen on which VAXTPU is running.

SCREEN

Specifies the coordinate system to be that used by the DECwindows window associated with VAXTPU's top-level widget.

window

Specifies the coordinate system to be that used by the VAXTPU window.

CHARACTERS

Specifies a system that measures screen distances in rows and columns, as a character-cell terminal does. In a character-based system, the cell in the top row and the leftmost column has the coordinates (1,1).

COORDINATES

Specifies a DECwindows coordinate system in which coordinate units correspond to pixels. The pixel in the upper left corner has the coordinates (0, 0).

from_x_integer
from_y_integer

Integer values representing a point in the original coordinate system and units.

to_x_integer
to_y_integer

Variables of type integer representing a point in the specified coordinate system and units. Note that the previous contents of the parameters is deleted when VAXTPU places the resulting values in them. You must specify VAXTPU variables for the parameters *to_x_integer* and *to_y_integer*. Passing a constant integer, string or keyword value, causes an error. (This requirement does not apply to the parameters *from_x_integer* and *from_y_integer*.)

DESCRIPTION

The converted coordinates are returned using the *to_x_integer* and *to_y_integer* parameters. Note that coordinate systems are distinguished both by units employed and where each places its origin.

SIGNALED ERRORS

TPU$_ARGMISMATCH	ERROR	The data type of the indicated parameter is not supported by the built-in.
TPU$_BADDELETE	ERROR	You are attempting to modify an integer, keyword, or string constant.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.
TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_WINDNOTVIS	WARNING	CONVERT cannot operate on an invisible window.

EXAMPLE

The following example converts a point's location from the current window's coordinate system (with the origin in the upper left-hand corner of the window) to the VAXTPU screen's coordinate system (with the origin in the upper left-hand corner of the VAXTPU screen). For more information about the difference between a VAXTPU window and the VAXTPU screen, see Chapter 6. If the current window is not the top window, CONVERT changes the value of the y-coordinate to reflect the difference in the VAXTPU screen's coordinate system.

```
PROCEDURE user_convert

LOCAL source_x,
      source_y,
      dest_x,
      dest_y;
```

```
source_x := 1;
source_y := 1;
dest_x := 0;
dest_y := 0;
CONVERT (CURRENT_WINDOW, COORDINATES, source_x, source_y, SCREEN, COORDINATES,
         dest_x, dest_y);
ENDPROCEDURE;
```

For another example of a procedure using the CONVERT built-in, see Example 7–1.

CREATE_RANGE

Continues to behave as it did in the version of VAXTPU released with VMS Version 5.0, except that the *video_attribute* parameter is now optional.

For more information on the CREATE_RANGE built-in and more examples of its use, see the *VAX Text Processing Utility Manual*.

FORMAT

range := CREATE_RANGE *(start_mark, end_mark [[, video_attribute]])*

PARAMETERS

start_mark
The starting mark for the range.

end_mark
The ending mark for the range.

video_attribute
A keyword designating the new video attribute for the range. The attribute can be NONE, REVERSE, UNDERLINE, BLINK, or BOLD. If you do not specify the parameter, the default is NONE.

return value

The range that has been created.

DESCRIPTION

The syntax of the CREATE_RANGE built-in has been changed to make CREATE_RANGE more consistent with MODIFY_RANGE. The video attribute of the CREATE_RANGE built-in is now optional. If not specified, the attribute defaults to NONE.

SIGNALED ERRORS

TPU$_NOTSAMEBUF	WARNING	First and second markers are in different buffers.
TPU$_TOOFEW	ERROR	CREATE_RANGE requires three parameters.
TPU$_TOOMANY	ERROR	CREATE_RANGE accepts no more than three parameters.
TPU$_NEEDTOASSIGN	ERROR	CREATE_RANGE must appear on the right-hand side of an assignment statement.
TPU$_INVPARAM	ERROR	One of your arguments to CREATE_RANGE is of the wrong type.
TPU$_BADKEY	WARNING	You specified an illegal keyword.

EXAMPLE

The following statement creates a range with no video attributes:

```
the_range := CREATE_RANGE (first_marker, last_marker);
```

GET_INFO (COMMAND_LINE, "character")

Fetches information about the character position specified when VAXTPU was invoked with the EDIT/TPU command.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(COMMAND_LINE, "character")*

PARAMETERS

COMMAND_LINE

A keyword directing VAXTPU to return information about what qualifiers and values were specified when VAXTPU was invoked.

"character"

A string indicating that the built-in is to return the column number of the character position specified by the /START_POSITION command qualifier.

DESCRIPTION

Returns the column number of the character position specified by the /START_POSITION command qualifier. This built-in is useful in a procedure to determine where VAXTPU should place the cursor at startup time. The default is 1 if the /START_POSITION qualifier is not specified.

This built-in is a synonym for GET_INFO (COMMAND_LINE, "start_character"). For more information about the /START_POSITION qualifier, see the *VAX Text Processing Utility Manual.*

GET_INFO (COMMAND_LINE, "line")

Fetches information about the record number specified when VAXTPU was invoked with the EDIT/TPU command.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(COMMAND_LINE, "line")*

PARAMETERS

COMMAND_LINE

A keyword directing VAXTPU to return information about what qualifiers and values were specified when VAXTPU was invoked.

"line"

A string indicating that the built-in is to return the record number of the line specified by the /START_POSITION command qualifier.

DESCRIPTION

Returns the record number of the line specified by the /START_POSITION command qualifier. This built-in is useful in a procedure to determine where VAXTPU should place the cursor at startup time. The default is 1 if the /START_POSITION qualifier is not specified.

This built-in is a synonym for GET_INFO (COMMAND_LINE, "start_record"). For more information about the /START_POSITION qualifier, see the *VAX Text Processing Utility Manual.*

GET_INFO (integer_variable, "name")

Fetches information about the specified integer or keyword variable.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

string := GET_INFO ({ *keyword,* / *integer,* } *"name")*

PARAMETERS

keyword

A VAXTPU keyword whose string equivalent you want. Note that in VAXTPU, key names are of type KEYWORD. Therefore, you can use this parameter to specify a key name whose string equivalent you want.

integer

The integer equivalent of a VAXTPU keyword whose string equivalent you want. When coding in VAXTPU, do not use an ordinal value obtained from this GET_INFO call as a substitute for the corresponding keyword. DIGITAL does not guarantee that the existing equivalences between integers and keywords will always remain the same.

"name"

A string requesting GET_INFO to return the string equivalent of the specified integer or keyword.

return value

string	The string representation of the specified keyword or integer equivalent of a keyword.

DESCRIPTION

Returns the string representation of a keyword or the integer equivalent of a keyword. The built-in performs all the functions that it performed in the version of VAXTPU released with VMS Version 5.0. In this version, the built-in has been enhanced so that if a key name is created using several key modifier keywords, the built-in returns the string representation of all the keywords used to create the key name. For more information on creating key names, see the description of the KEY_NAME built-in.

EXAMPLES

For an explanation of the VAXTPU code in these examples, see the paragraph following each numbered code example.

GET_INFO (integer_variable, "name")

1

```
new_key := KEY_NAME (KP4, SHIFT_MODIFIED, CTRL_MODIFIED);
!    .
!    .
!    .
IF GET_INFO (new_key, "key_modifiers")
THEN
    the_name := GET_INFO (new_key, "name")
ENDIF;
MESSAGE (STR (the_name));
```

The first statement in this code fragment creates a VAXTPU key name for the key sequence produced by pressing the Ctrl key, the Shift key, and the 4 key on the keypad, all at once. The new key name is assigned to the variable *key_name*. The IF clause of the statement shows how to test whether a key name was created using one or more key modifier keywords. (Note, however, that this statement would not detect whether a key name was created using the keyword Shift_key. The built-in GET_INFO (key_name, "key_modifiers") returns 0 even if the key name was created using SHIFT_KEY.) The THEN clause shows how to fetch the key modifier keyword or keywords used to create a key name. The final statement displays the string KEY_NAME (KP4, SHIFT_MODIFIED, ALT_MODIFIED) in the message buffer.

2

```
equiv_string := GET_INFO (14, "name");
```

This statement assigns the string *object* to the variable *equiv_string*. The value 14 is the integer equivalent of the keyword OBJECT.

GET_INFO (marker_variable, "record_number")

Fetches information about the specified marker.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(marker_variable, "record_number")*

PARAMETERS

marker_variable
The marker whose location you want the built-in to return.

"record_number"
A string requesting GET_INFO to return the number associated with the record containing the specified marker.

return value

The number associated with the record containing the specified marker.

DESCRIPTION

This built-in returns the number associated with the record containing the specified marker.

A record number indicates the location of a record in a buffer. Record numbers are dynamic; as you add or delete records, VAXTPU changes the number associated with a particular record, as appropriate. VAXTPU counts each record in a buffer, regardless of whether the line is visible in a window, or whether the record contains text.

EXAMPLE

The following statements establish a marker at the editing point, then fetch the number associated with the record where the marker is located:

```
here := MARK (FREE_CURSOR);
wheres_here := GET_INFO (here, "record_number");
```

GET_INFO (mouse_event_keyword, "mouse_button")

Fetches information about a mouse event.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual*.

FORMAT

integer := GET_INFO *(mouse_event_keyword, "mouse_button")*

PARAMETERS

mouse_event_keyword

A keyword representing a single click, multiple click, upstroke, downstroke, or drag of a mouse button. For a list of the valid keywords, see Table 4–1.

"mouse_button"

A string requesting GET_INFO to return the number of the specified mouse button.

return value

The number of the mouse button specified with the *mouse_event_keyword* parameter.

DESCRIPTION

Table 4–1 lists the valid keywords for this built-in.

Table 4–1 VAXTPU Keywords Representing Mouse Events

M1UP	M2UP	M3UP	M4UP	M5UP
M1DOWN	M2DOWN	M3DOWN	M4DOWN	M5DOWN
M1DRAG	M2DRAG	M3DRAG	M4DRAG	M5DRAG
M1CLICK	M2CLICK	M3CLICK	M4CLICK	M5CLICK
M1CLICK2	M2CLICK2	M3CLICK2	M4CLICK2	M5CLICK2
M1CLICK3	M2CLICK3	M3CLICK3	M4CLICK3	M5CLICK3
M1CLICK4	M2CLICK4	M3CLICK4	M4CLICK4	M5CLICK4
M1CLICK5	M2CLICK5	M3CLICK5	M4CLICK5	M5CLICK5

EXAMPLES

For an explanation of the VAXTPU code in these examples, see the paragraph following each numbered code example.

```
1   x := GET_INFO (M3CLICK2, "mouse_button");
```

This statement causes VAXTPU to assign the value 3 to the variable *x*.

```
2   the_key := READ_KEY;
    IF GET_INFO (the_key, "mouse_button") = 3
    THEN
        MESSAGE ("MB3 has no effect in this context.");
```

These statements test whether you have pressed MB3 and, if so, displays a message in the message window.

GET_INFO (mouse_event_keyword, "window")

Fetches the window in which the down stroke occurred that started the current drag operation.

FORMAT

{ integer / window } := GET_INFO *(mouse_event_keyword, "window")*

PARAMETERS

mouse_event_keyword

A keyword representing the down stroke of one of the mouse buttons. The valid keywords for this parameter are M1DOWN, M2DOWN, M3DOWN, M4DOWN, and M5DOWN.

"window"

A string requesting GET_INFO to return the window in which the down stroke occurred that started the current drag operation.

return value

integer	The value 0, which is returned if no drag operation is in progress for the specified mouse button when the built-in is executed.
window	The window in which the down click occurred that started the current drag operation.

EXAMPLE

The following procedure, when bound to M1DRAG, responds to a drag event by checking whether you have dragged the mouse across window bounds. If you have done so, the procedure displays a message. If not, the procedure creates a select range.

```
PROCEDURE sample_m1_drag

LOCAL the_window,
      new_window,
      column,
      row,
      temp;

the_window := GET_INFO (M1DOWN, "window");
IF the_window = 0
THEN
    RETURN (FALSE)
ENDIF;

LOCATE_MOUSE (new_window, column, row);
```

```
IF the_window <> new_window
THEN
    MESSAGE ("Invalid drag of pointer across window boundaries.");
ENDIF;
ENDPROCEDURE;
```

GET_INFO (SCREEN, "decwindows")

Indicates whether VAXTPU is using its DECwindows screen updater.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(SCREEN, "decwindows")*

PARAMETERS

SCREEN

A keyword indicating that GET_INFO is to fetch information about the VAXTPU screen.

"decwindows"

A string requesting GET_INFO to indicate whether VAXTPU is using its DECwindows screen updater.

return value

Returns 1 if your system is running the DECwindows version of VAXTPU, otherwise 0.

GET_INFO (SCREEN, "line_editing")

On a character-cell terminal, indicates whether the line editing terminal attribute is turned on.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual*.

FORMAT

integer := GET_INFO *(SCREEN, "line_editing")*

PARAMETERS

SCREEN

A keyword indicating that GET_INFO is to fetch information about the VAXTPU screen.

"line_editing"

A string requesting GET_INFO to indicate whether the line editing terminal attribute is turned on.

return value

On a character-cell terminal, returns 1 if the line editing terminal attribute is turned on, otherwise returns 0. In DECwindows VAXTPU, this built-in always returns 0.

GET_INFO (SCREEN, "original_length")

Returns the number of lines the screen had when VAXTPU was started.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(SCREEN, "original_length")*

PARAMETERS

SCREEN

A keyword indicating that GET_INFO is to fetch information about the VAXTPU screen.

"original_length"

A string indicating that GET_INFO is to fetch the number of lines the screen had when VAXTPU was started.

return value

The number of lines the screen had when VAXTPU was started.

GET_INFO (SYSTEM, "timer")

Indicates whether the built-in SET (TIMER) has been enabled.

For more information about the SET (TIMER) built-in and the GET_INFO built-in, see the *VAX Text Processing Utility Manual*.

FORMAT

integer := GET_INFO *(SYSTEM, "timer")*

PARAMETERS

SYSTEM

A keyword indicating that GET_INFO is to fetch information about global settings in VAXTPU.

"timer"

A string indicating that GET_INFO is to indicate whether the SET (TIMER) built-in is enabled.

return value

The value 1 if the SET (TIMER) built-in is enabled, otherwise 0.

GET_INFO (window_variable, "bottom")

Fetches the number of the last row or last visible row of the specified window, or the specified window's text area.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(window_variable, "bottom"* [, *WINDOW* | , *TEXT* | , *VISIBLE_WINDOW* | , *VISIBLE_TEXT*])

PARAMETERS

window_variable

The window for which you want the specified row.

"bottom"

A string requesting GET_INFO to fetch the number of the last row or last visible row of the specified window or the specified window's text area.

TEXT

A keyword directing the built-in to return the last window row on which text can be displayed. By specifying TEXT instead of VISIBLE_TEXT, you obtain the window's last text row even if that row is invisible because the window is occluded. If the window is not occluded, the value returned is the same as the value returned with VISIBLE_TEXT. GET_INFO (window, "bottom", TEXT) is a synonym for GET_INFO (window, "original_bottom").

VISIBLE_TEXT

A keyword directing the built-in to return the number of the last visible window row on which text can be displayed. The call GET_INFO (window, "bottom", VISIBLE_TEXT) is a synonym for GET_INFO (window, "visible_bottom"). When VAXTPU determines a window's last visible text row for either of these built-ins, VAXTPU does not consider the status line or the bottom scroll bar to be a text row.

VISIBLE_WINDOW

A keyword directing the built-in to return the number of the last visible row in the window. There is no synonym for this call.

WINDOW

A keyword directing the built-in to return the number of the last row in the window. By specifying WINDOW instead of TEXT, you obtain the window's last row, even if the last row cannot contain text because it contains a scroll bar or status line. By specifying WINDOW instead of VISIBLE_WINDOW, you obtain the window's last row even if that row is invisible because the window is occluded. If the window is not occluded, the value returned is the same as the value returned with VISIBLE_

WINDOW. GET_INFO (window, "bottom", WINDOW) is a synonym for GET_INFO ("original_bottom").

return value

integer — The number of the last row or last visible row.

DESCRIPTION

The window row whose number is returned depends on the keyword you specify. If you do not specify a keyword, the default is TEXT.

EXAMPLE

The following statement returns the last line of the window "bottom_ window." The value returned is the line containing the status line or scroll bar, whichever comes last, if the window has a status line or scroll bar.

```
last_line := GET_INFO (bottom_window, "bottom", WINDOW);
```

For another example of code using GET_INFO (window_variable, "bottom", WINDOW) see Example 7–5.

GET_INFO (window_variable, "key_map_list")

Fetches the string that is the name of the key map list associated with the window you specify.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

string := GET_INFO *(window_variable, "key_map_list")*

PARAMETERS

window_variable

The window for which you want the associated key map list.

"key_map_list"

A string directing GET_INFO to fetch the key map list for the specified window.

return value

string	The name of the key map list associated with the specified window.

EXAMPLE

The following statement returns the key map list associated with the current window:

```
current_list := GET_INFO (CURRENT_WINDOW, "key_map_list");
```

For an example of code using GET_INFO (window_variable, "key_map_list", WINDOW) see Example 7–6.

GET_INFO (window_variable, "left")

Fetches the number of the leftmost column or leftmost visible column of the specified window, or the specified window's text area.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(window_variable, "left"*
[, WINDOW
[, TEXT
[, VISIBLE_WINDOW
[, VISIBLE_TEXT *]]]])*

PARAMETERS

window_variable
The window for which you want the leftmost column.

"left"
A string requesting GET_INFO to fetch the number of the leftmost column or leftmost visible column of the specified window or the specified window's text area.

TEXT
A keyword directing the built-in to return the first window column in which text can be displayed. By specifying TEXT instead of VISIBLE_TEXT, you obtain the window's first text column, even if that column is invisible because the window is occluded. If the window is not occluded, the value returned is the same as the value returned with VISIBLE_TEXT. In Version 2.2, the value returned with TEXT is also the same as the value returned with WINDOW.

VISIBLE_TEXT
A keyword directing the built-in to return the number of the first visible window column in which text can be displayed.

VISIBLE_WINDOW
A keyword directing the built-in to return the number of the first visible column in the window.

WINDOW
A keyword directing the built-in to return the number of the first column in the window. By specifying WINDOW instead of VISIBLE_WINDOW, you obtain the window's first column even if that column is invisible because the window is occluded. If the window is not occluded, the value returned is the same as the value returned with VISIBLE_WINDOW.

return value

Integer	The number of the leftmost column or leftmost visible column.

DESCRIPTION

The column whose number is returned depends on the keyword you specify. If you do not specify a keyword, the default is TEXT.

EXAMPLE

The following statement returns the leftmost column where text can be displayed in the current window. Note that changing the left margin setting has no effect on the value returned.

```
first_column := GET_INFO (CURRENT_WINDOW, "left", TEXT);
```

GET_INFO (window_variable, "length")

Fetches the number of rows or visible rows in the specified window or the specified window's text area.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(window_variable, "length"* [, *WINDOW* | , *TEXT* | , *VISIBLE_WINDOW* | , *VISIBLE_TEXT*] *)*

PARAMETERS

window_variable

The window for which you want the specified number of rows.

"length"

A string requesting GET_INFO to fetch the number of rows or visible rows in the specified window or the specified window's text area.

TEXT

A keyword directing the built-in to return the number of window rows on which text can be displayed. By specifying TEXT instead of VISIBLE_TEXT, you obtain the total number of window text rows even if some of them are invisible because the window is occluded. If the window is not occluded, the value returned is the same as the value returned with VISIBLE_TEXT.

VISIBLE_TEXT

A keyword directing the built-in to return the number of visible window rows on which text can be displayed. There is no synonym for this call.

VISIBLE_WINDOW

A keyword directing the built-in to return the number of visible rows in the window. GET_INFO (window, "length", VISIBLE_WINDOW) is a synonym for GET_INFO (window, "visible_length"). When VAXTPU calculates the visible text rows for either of these calls, VAXTPU does not consider the status line or the horizontal scroll bar to be a text row.

WINDOW

A keyword directing the built-in to return the number of rows in the window. By specifying WINDOW instead of TEXT, you obtain the total number of window rows even if the last rows cannot contain text because they contain a scroll bar or status line. By specifying WINDOW instead of VISIBLE_WINDOW, you obtain the total number of window rows even if some rows are invisible because the window is occluded. If the window is not occluded, the value returned is the same as the value returned with VISIBLE_WINDOW. GET_INFO (window, "length", WINDOW) is a synonym for the call GET_INFO (window, "original_length").

GET_INFO (window_variable, "length")

return value

integer	The number of rows or the number of visible rows.

DESCRIPTION

The number of rows returned depends on the keyword you specify. If you do not specify a keyword, the default is TEXT.

EXAMPLE

The following statement adds the length of the window (the value in *the_window*) to the value in *the_length*. Note that the length of the window includes the length added by the scroll bar and status line, if the window has them.

```
the_length := the_length + GET_INFO (the_window, "length", WINDOW);
```

For another example of code using GET_INFO (window_variable, "length", WINDOW) see Example 7–5.

GET_INFO (window_variable, "right")

Fetches the number of the last column or last visible column of the specified window or the specified window's text area.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(window_variable, "right"*
[[, WINDOW
, TEXT
, VISIBLE_WINDOW]])
, VISIBLE_TEXT

PARAMETERS

window_variable
The window for which you want the specified row.

"right"
A string requesting GET_INFO to fetch the number of the last column or last visible column of the specified window or the specified window's text area.

TEXT
A keyword directing the built-in to return the last window column in which text can be displayed. By specifying TEXT instead of VISIBLE_TEXT, you obtain the window's last text column even if that column is invisible because the window is occluded. If the window is not occluded, the value returned is the same as the value returned with VISIBLE_TEXT.

VISIBLE_TEXT
A keyword directing the built-in to return the number of the last visible window column in which text can be displayed.

VISIBLE_WINDOW
A keyword directing the built-in to return the number of the last visible row in the window.

WINDOW
A keyword directing the built-in to return the number of the last column in the window. By specifying WINDOW instead of TEXT, you obtain the window's last column even if the last column cannot contain text because it contains a scroll bar.

return value

integer	The number of the last column or last visible column.

DESCRIPTION

The window column whose number is returned depends on the keyword you specify. If you do not specify a keyword, the default is TEXT.

EXAMPLE

The following statement returns the number of the rightmost column in the current window. Note that the column whose number is returned can be occupied by a vertical scroll bar if one is present. Note, too, that the returned value changes if you widen the window, but not if you move the window without widening it.

```
last_column := GET_INFO (CURRENT_WINDOW, "right", WINDOW);
```

GET_INFO (window_variable, "top")

Fetches the number of the first row or first visible row of the specified window or the specified window's text area.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(window_variable, "top"* [, *WINDOW* | , *TEXT* | , *VISIBLE_WINDOW* | , *VISIBLE_TEXT*] *)*

PARAMETERS

window_variable
The window for which you want the specified row.

"top"
A string requesting GET_INFO to fetch the number of the first row or first visible row of the specified window or the specified window's text area.

TEXT
A keyword directing the built-in to return the first window row on which text can be displayed. By specifying TEXT instead of VISIBLE_TEXT, you obtain the window's first text row even if that row is invisible because the window is occluded. If the window is not occluded, the value returned is the same as the value returned with VISIBLE_TEXT. In Version 2.2, the value returned by TEXT is also the same as the value returned by WINDOW. GET_INFO (window_variable, "top", TEXT) is a synonym for GET_INFO (window_variable, "original_top").

VISIBLE_TEXT
A keyword directing the built-in to return the number of the first visible window row on which text can be displayed. GET_INFO (window_variable, "top", VISIBLE_TEXT) is a synonym for GET_INFO (window_variable, "visible_top").

VISIBLE_WINDOW
A keyword directing the built-in to return the number of the first visible row in the window.

WINDOW
A keyword directing the built-in to return the number of the first row in the window. By specifying WINDOW instead of VISIBLE_WINDOW, you obtain the window's first row even if that row is invisible because the window is occluded. If the window is not occluded, the value returned is the same as the value returned with VISIBLE_WINDOW. GET_INFO (window_variable, "top", WINDOW) is a synonym for GET_INFO ("original_top").

return value

integer	The number of the first row or first visible row.

DESCRIPTION

The window row whose number is returned depends on the keyword you specify. If you do not specify a keyword, the default is TEXT.

EXAMPLE

The following statement returns the number of the first row in the current window. Note that the row number returned is relative to the top of the VAXTPU screen. Thus, if the current window is not the top window on the VAXTPU screen, the row number returned is not 1.

```
first_row := GET_INFO (CURRENT_WINDOW, "top", WINDOW);
```

For another example of code using GET_INFO (window_variable, "top", WINDOW) see Example 7–5.

GET_INFO (window_variable, "width")

Fetches the number of columns or the number of visible columns in the specified window or the specified window's text area.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(window_variable, "width"* [, *WINDOW* | , *TEXT* | , *VISIBLE_WINDOW* | , *VISIBLE_TEXT*])

PARAMETERS

window_variable

The window for which you want the specified row.

"width"

A string requesting GET_INFO to fetch the number of columns or visible columns in the specified window or the specified window's text area.

TEXT

A keyword directing the built-in to return the number of window columns in which text can be displayed. By specifying TEXT instead of VISIBLE_TEXT, you obtain the total number of window text columns even if some of them are invisible because the window is occluded. If the window is not occluded, the value returned is the same as the value returned with VISIBLE_TEXT. GET_INFO (window_variable, "width", TEXT) is a synonym for GET_INFO (window_variable, "width").

VISIBLE_TEXT

A keyword directing the built-in to return the number of visible window columns on which text can be displayed. There is no synonym for this call.

VISIBLE_WINDOW

A keyword directing the built-in to return the number of visible window columns in which text can be displayed.

WINDOW

A keyword directing the built-in to return the number of columns in the window. By specifying WINDOW instead of TEXT, you obtain the total number of window columns even if the last column cannot contain text because it contains a scroll bar. By specifying WINDOW instead of VISIBLE_WINDOW, you obtain the total number of window columns even if some columns are invisible because the window is occluded. If the window is not occluded, the value returned is the same as the value returned with VISIBLE_WINDOW.

return value

integer	The number of columns or the number of visible columns.

DESCRIPTION

The number of columns returned depends on the keyword you specify. If you do not specify a keyword, the default is TEXT.

EXAMPLE

The following statement returns the number of columns in the current window:

```
the_width := GET_INFO (CURRENT_WINDOW, "width", WINDOW);
```

For an example of code using GET_INFO (window_variable, "width", WINDOW) see Example 7–6.

INT

Accepts an integer as a parameter without generating an error.

The INT built-in is supported in previous versions of VAXTPU. INT continues to perform all the functions it performed in previous versions; the addition of the data type INTEGER as a valid data type for the parameter is new in DECwindows VAXTPU.

For more information about the INT built-in, see the *VAX Text Processing Utility Manual*.

FORMAT

integer2 := INT *(integer1)*

PARAMETERS

integer1
Any integer value. INT accepts a parameter of type INTEGER so you need not check the type of the parameter you supply to the built-in.

return value

The integer equivalent of the parameter you specify.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	INT returns a value that must be used.
TPU$_TOOFEW	ERROR	INT requires one parameter.
TPU$_TOOMANY	ERROR	INT accepts only one parameter.
TPU$_ARGMISMATCH	ERROR	The parameter to INT was not a keyword or string.
TPU$_INVNUMSTR	WARNING	The string you passed to INT was not a number.
TPU$_NULLSTRING	WARNING	You passed a string of length 0 to INT.

EXAMPLE

The following statement assigns the integer 9 to the variable *the_int*:

```
the_int := INT (9);
```

LOCATE_MOUSE

Returns the current position of the mouse if VAXTPU finds the mouse in a VAXTPU window.

FORMAT

[integer :=] LOCATE_MOUSE *[(window, x_integer, y_integer)]*

PARAMETERS

window

The window in which the pointer cursor is located. If the pointer cursor was not in a VAXTPU window at the time of the most recent keyboard or mouse event, this parameter is assigned the type UNSPECIFIED.

x_integer

The number of the window column where the pointer cursor is located. If the pointer cursor was not in a VAXTPU window at the time of the most recent keyboard or mouse event, this parameter is assigned the type UNSPECIFIED.

y_integer

The number of the window row where the pointer cursor is located. If the pointer cursor was located on a window's status line at the time of the most recent keyboard or mouse event, this parameter is assigned the value 0. If the pointer cursor was not in a VAXTPU window at the time of the most recent keyboard or mouse event, this parameter is assigned the type UNSPECIFIED.

return value

integer	The value 1 if VAXTPU finds a window position, otherwise 0.

DESCRIPTION

In the DECwindows version of VAXTPU, you can use the built-in LOCATE_MOUSE anytime after the first keyboard or mouse-button event. The built-in returns the location occupied by the pointer cursor at the time of the most recent keyboard or mouse button event.

If there is no mouse information available (because no mouse button has been pressed or if the mouse has been disabled using SET (MOUSE)), LOCATE_MOUSE signals the status TPU$_MOUSEINV.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	LOCATE_MOUSE requires three parameters.
TPU$_TOOMANY	ERROR	LOCATE_MOUSE accepts at most three parameters.

TPU$_BADDELETE	ERROR	You have specified a constant as one or more of the parameters.
TPU$_MOUSEINV	WARNING	No mouse button has been pressed or the mouse has been disabled.

EXAMPLE

The following statement returns an integer in the variable *status* indicating whether the pointer cursor was found in a window, the window in the parameter *new_window* where the mouse was found, an integer in the parameter *x_value* specifying the pointer cursor's location in the horizontal dimension, and an integer in the parameter *y_value* specifying the pointer cursor's location in the vertical dimension.

```
status := LOCATE_MOUSE (new_window, x_value, y_value);
```

To see the LOCATE_MOUSE built-in used in a procedure, see Example 7–1 and Example 7–9.

MODIFY_RANGE

Supports dynamic alteration of a range.

FORMAT

MODIFY_RANGE *(range, [[mark1, mark2]] [[, video_attribute]])*

PARAMETERS

range
The range that is to be modified.

mark1
A marker delimiting one end of the range.

mark2
A marker delimiting the other end of the range.

video_attribute
A keyword designating the new video attribute for the range. The attribute can be NONE, REVERSE, UNDERLINE, BLINK, or BOLD. If not specified, the video attribute for the range remains the same.

DESCRIPTION

You can use MODIFY_RANGE to specify a new starting mark and ending mark for an existing range. The new ending mark can be above the new starting mark in the buffer.

MODIFY_RANGE can also change the characteristics of the range without deleting, re-creating, and repainting all the characters in the range. Using MODIFY_RANGE, you can direct VAXTPU to apply or remove the range's video attribute to or from characters as you select and unselect text.

Ranges are limited to one video attribute at a time. Specifying a video attribute different than the present attribute causes VAXTPU to apply the new attribute to the entire visible portion of the range.

If the video attribute stays the same and only the markers move, the only characters that are refreshed are those visible characters newly added to the range and those visible characters that are no longer part of the range.

SIGNALED ERRORS

TPU$_NOTSAMEBUF	WARNING	First and second markers are in different buffers.
TPU$_ARGMISMATCH	ERROR	The data type of the indicated parameter is not supported by the built-in.
TPU$_BADKEY	WARNING	You specified an illegal keyword.

TPU$_INVPARAM	ERROR	You specified a parameter of the wrong type.
TPU$_MODRANGEMARKS	ERROR	MODIFY_RANGE requires either two marker parameters or none.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.

EXAMPLES

For an explanation of the VAXTPU code in these examples, see the paragraph following each numbered code example.

1
```
begin_mark := MARK (BOLD);
POSITION (MOUSE);
finish_mark := MARK (BOLD);
this_range := CREATE_RANGE (begin_mark, finish_mark, BOLD);
!  .
!  .  (User may have moved mouse)
!  .
POSITION (MOUSE);
new_mark := MARK (BOLD);
IF new_mark <> finish_mark
THEN
    MODIFY_RANGE (this_range, begin_mark, new_mark, BOLD);
ENDIF;
```

This code fragment creates a range between the editing point and the pointer cursor location. At a later point in the program, after which you might have moved the pointer cursor, the code fragment modifies the range to reflect the new pointer cursor location.

2
```
MODIFY_RANGE (this_range, , ,BLINK);
```

This statement sets the video attribute of the range *this_range* to BLINK. Notice that you must use commas as placeholders if you do not specify the starting and ending marks of the range.

3
```
PROCEDURE move_mark (place_to_start, direction);
    POSITION (place_to_start);
    IF direction = 1
    THEN
        MOVE_HORIZONTAL (1);
    ELSE
        MOVE_HORIZONTAL (-1);
    ENDIF;
    RETURN MARK (NONE);
ENDPROCEDURE;
PROCEDURE user_shrink_and_enlarge_range
```

```
    LOCAL  start_mark,
           end_mark,
           direction,
           dynamic_range,
           rendition,
           remembered_range;

                                          ! The following lines
                                          ! create a range that
                                          ! shrinks and grows and
                                          ! a range that defines
                                          ! the limits of the dynamic
                                          ! range.

    POSITION (LINE_BEGIN);
    start_mark := MARK (NONE);
    POSITION (LINE_END);
    end_mark := MARK (NONE);
    rendition := REVERSE;
    remembered_range := CREATE_RANGE (start_mark, end_mark, NONE);
    dynamic_range := CREATE_RANGE (start_mark, end_mark, rendition);

                                           ! The following lines
                                           ! shrink and enlarge
                                           ! the dynamic range.

    direction := 1;

    LOOP
        UPDATE (CURRENT_WINDOW);

        start_mark := move_mark (BEGINNING_OF (dynamic_range), direction);
        end_mark := move_mark (END_OF (dynamic_range), 1 - direction);

        MODIFY_RANGE (dynamic_range, start_mark, end_mark);

        IF start_mark > end_mark
        THEN
            EXITIF READ_KEY = CTRL_Z_KEY;
            direction := 0;
            IF rendition = REVERSE
            THEN
                rendition := BOLD;
            ELSE
                rendition := REVERSE;
            ENDIF;
            MODIFY_RANGE (dynamic_range, , , rendition);

        ENDIF;

        IF (start_mark = BEGINNING_OF (remembered_range)) OR
           (end_mark = END_OF (remembered_range))
        THEN
            direction := 1;
        ENDIF;
    ENDLOOP;

ENDPROCEDURE;
```

These procedures cause the range *dynamic_range* to shrink to one character, then grow until it becomes as large as the range *remembered_range*.

```
4   PROCEDURE line_up_characters (text_range, lined_chars_pat)
    LOCAL
        range_start,
        range_end,
        temp_range,
        max_cols;


    range_end := END_OF (text_range);            ! These statements store
                                                 ! the ends of the range
                                                 ! containing the text operated on
    range_start := BEGINNING_OF (text_range);

                                              ! The following statements
                                              ! locate the portions of
                                              ! text that match the pattern
                                              ! and determine which is
                                              ! furthest to the right.
    max_cols := 0;
    LOOP
        temp_range := SEARCH_QUIETLY (lined_chars_pat, REVERSE, EXACT, text_range);
        EXITIF temp_range = 0;
        POSITION (temp_range);
        IF GET_INFO (MARK (NONE), "offset_column") > max_cols
        THEN
            max_cols := GET_INFO (MARK (NONE), "offset_column");
        ENDIF;
        MOVE_HORIZONTAL (-1);
        MODIFY_RANGE (text_range, BEGINNING_OF (text_range), MARK (NONE));
    ENDLOOP;

                                                    ! The following lines
                                                    ! locate matches to the
    text_range := CREATE_RANGE (range_start, range_end);  ! pattern and align them
                                                    ! with the rightmost
                                                    ! piece of matching text.
    LOOP
        temp_range := SEARCH_QUIETLY (lined_chars_pat, FORWARD, EXACT, text_range);
        EXITIF temp_range = 0;
        POSITION (temp_range);
        IF GET_INFO (MARK (NONE), "offset_column") < max_cols
        THEN
            COPY_TEXT (" " * (max_cols - GET_INFO (MARK (NONE), "offset_column")));
        ENDIF;
        MOVE_HORIZONTAL (1);
        MODIFY_RANGE (text_range, END_OF (text_range), MARK (NONE));
    ENDLOOP;

    !
    ! Restore the range to its original state, plus a reverse attribute.
    !
    text_range := CREATE_RANGE (range_start, range_end, REVERSE); ! This line
                                                                  ! restores the
                                                                  ! range to its
                                                                  ! original state
                                                                  ! and displays
                                                                  ! the contents
                                                                  ! in reverse video.

    ENDPROCEDURE;
```

MODIFY_RANGE

This procedure aligns text that conforms to the pattern specified in the second parameter. For example, if you want to align all comments in a piece of VAXTPU code, you would pass as the second parameter a pattern defined as an exclamation point followed by an arbitrary amount of text or white space and terminated by a line end.

The procedure is passed a range of text. As the procedure searches the range to identify the rightmost piece of text that matches the pattern, the procedure modifies the range to exclude any matching text. Next, the procedure searches the original range again and inserts padding spaces in front of each instance of matching text, making the text align with the rightmost instance of matching text.

POSITION (integer)

Moves the editing point to the specified record in the current buffer.

The POSITION built-in is supported in previous versions of VAXTPU. POSITION continues to perform all the functions it performed in previous versions; the use of POSITION to move the editing point to a specified record is new in the current version of VAXTPU.

For more information about the POSITION built-in, see the *VAX Text Processing Utility Manual*.

FORMAT

POSITION *(integer)*

PARAMETERS

integer
The number of the record where you want VAXTPU to position the editing point.

DESCRIPTION

A record number indicates the location of a record in a buffer. Record numbers are dynamic; as you add or delete records, VAXTPU changes the number associated with a particular record, as appropriate. VAXTPU counts each record in a buffer, regardless of whether the line is visible in a window, or whether the record contains text.

To position the editing point to a given record, specify the record number. The number can be in the range from 1 to the number of records in the buffer plus 1. For example, the following statement positions the editing point to record number 8 in the current buffer:

```
POSITION (8);
```

VAXTPU places the editing point on the first character of the record.

Specifying a value of 0 has no effect. Specifying a negative number or a number greater than the number of records in the buffer plus 1 causes VAXTPU to signal an error.

VAXTPU places the editing point on the first character in the record.

SIGNALED ERRORS

TPU$_ARGMISMATCH	ERROR	Wrong type of data sent to the built-in.
TPU$_BADVALUE	ERROR	You specified a record number less than 0 or greater than the length of the buffer plus 1.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.

TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following statement moves the editing point to the tenth line in the current buffer.

```
POSITION (10);
```

For an example of a procedure using the POSITION (integer) built-in, see Example 7–8.

POSITION (MOUSE)

Positions the editing point to the location indicated by the pointer cursor.

The POSITION built-in is supported in previous versions of VAXTPU. POSITION continues to perform all the functions it performed in previous versions; the fact that POSITION (MOUSE) can be used in contexts other than programs bound to mouse keys is new in DECwindows VAXTPU.

FORMAT

POSITION *(MOUSE)*

PARAMETERS

MOUSE
A keyword directing VAXTPU to associate the editing point with the location of the pointer cursor.

DESCRIPTION

In the DECwindows version of VAXTPU, you can use the statement POSITION (MOUSE) at any point after the first keyboard or mouse button event. The statement positions the editing point to the location occupied by the pointer cursor at the time of the most recent keyboard or mouse-button event.

If the pointer cursor is on a window's status line when POSITION (MOUSE) is executed, VAXTPU positions the editing point at the line just above the status line.

If the pointer cursor is not located in a VAXTPU window at the time of the most recent keyboard or mouse-button event, POSITION (MOUSE) returns the status TPU$_NOWINDOW.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.
TPU$_NOWINDOW	WARNING	The pointer cursor was not located in a VAXTPU window at the time of the most recent keyboard or mouse-button event.

SET (KEY_MAP_LIST)

Associates the named key map list with the specified window.

SET (KEY_MAP_LIST) is supported in previous versions of VAXTPU. The built-in continues to behave as it did in previous versions. The fact that you can associate a key map list with a window supplements the fact that you can associate a key map list with a buffer.

For more information about the SET (KEY_MAP_LIST) built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

SET *(KEY_MAP_LIST, string, window)*

PARAMETERS

KEY_MAP_LIST

A keyword used to establish a key map list.

string

The name of the key map list you want to associate with a window.

window

The window with which you want to associate the key map list.

DESCRIPTION

The key map list manipulated by this built-in is used only to process mouse events in the specified window. Keystrokes are processed with the key map list associated with the buffer.

SIGNALED ERRORS

TPU$_NOKEYMAPLIST	WARNING	Attempt to access an undefined key map list.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.
TPU$_NOCURRENTBUF	ERROR	You are not positioned in a buffer.
TPU$_INVPARAM	ERROR	Wrong type of data sent to the built-in.

EXAMPLE

The following procedure creates a small "scratch pad" window and maps it to a scratch buffer called *junk1.txt*. The procedure defines a key map list consisting of a user-define key map redefining M1DRAG plus the standard EVE mouse key map. By setting the scratch window's key map list to be *user_scratch_list*, the procedure invokes *sample_m1_drag* when the user drags the mouse in the scratch window.

```
PROCEDURE user_scratch_window

LOCAL scratch_window,
      scratch_buffer,
      scratch_map,
      scratch_list;

scratch_window := CREATE_WINDOW (20, 3, ON);
scratch_buffer := CREATE_BUFFER ("test", "junk.txt");
scratch_map := CREATE_KEY_MAP ("user_scratch_map");
DEFINE_KEY (eve$$kt_return + "sample_M1_DRAG", M1DRAG, "mouse_button_1",
            "user_scratch_map");
scratch_list := CREATE_KEY_MAP_LIST ("user_scratch_list", "user_scratch_map",
                                     eve$x_mouse_keys);
SET (KEY_MAP_LIST, "user_scratch_list", scratch_window);
MAP (scratch_window, scratch_buffer);

ENDPROCEDURE;
```

SET (MODIFIED)

Turns on or turns off the bit indicating that the specified buffer has been modified.

FORMAT

SET *(MODIFIED, buffer,* { *ON* / *OFF* } *)*

PARAMETERS

MODIFIED
A keyword directing VAXTPU to turn on or turn off the indicator designating a buffer as modified.

buffer
The buffer whose indicator you want to control.

ON
A keyword directing VAXTPU to mark a buffer as modified.

OFF
A keyword directing VAXTPU to mark a buffer as unmodified.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following statement marks the current buffer as modified:

```
SET (MODIFIED, CURRENT_BUFFER, ON);
```

SET (MOUSE)

Assigns the keyword ON or OFF to the return variable you specify.

The SET (MOUSE) built-in is supported in the version of VAXTPU released with VMS Version 5.0. SET (MOUSE) continues to perform all the functions it performed in previous versions; the return values ON and OFF are new in this version of VAXTPU.

For more information on the SET (MOUSE) built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

[{ **ON** / **OFF** } **:=]** **SET** *(MOUSE,* { *ON* / *OFF* } *)*

DESCRIPTION

The return value specifies whether VAXTPU mouse support was enabled or disabled before the current SET (MOUSE) statement was executed. This allows you to enable or disable mouse support and then reset the support to its previous setting without having to make a separate call.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	The keyword must be either ON or OFF.
TPU$_MOUSEINV	WARNING	You have tried to enable mouse support on an incompatible terminal.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_TOOFEW	ERROR	SET (MOUSE) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.

SET (SCREEN_UPDATE)

Returns a variable of type KEYWORD indicating whether screen updating was enabled or disabled before the current SET (SCREEN_UPDATE) statement was executed.

The SET (SCREEN_UPDATE) built-in is supported in the version of VAXTPU released with VMS Version 5.0. SET (SCREEN_UPDATE) continues to perform all the functions it performed in previous versions; the fact that it allows a return value of type KEYWORD is new in this version of VAXTPU.

For more information about this built-in, see the *VAX Text Processing Utility Manual*.

FORMAT

[{ **ON** / **OFF** } **:=]** **SET** *(SCREEN_UPDATE,* { *ON* / *OFF* } *)*

PARAMETERS

SCREEN_UPDATE

A keyword directing VAXTPU to set an attribute of screen updating.

ON

A keyword indicating that screen updating is enabled.

OFF

A keyword indicating that screen updating is disabled.

return value

A variable containing the keyword value ON or OFF. The keyword specifies whether VAXTPU screen updating support was enabled or disabled before the current SET (SCREEN_UPDATE) statement was executed. Using the returned variable, you can enable or disable screen updating and then reset the support to its previous setting without having to make a separate call to fetch the previous setting.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	The keyword must be ON or OFF.
TPU$_INVPARAM	ERROR	One or more of the specified parameters have the wrong type.
TPU$_TOOFEW	ERROR	SET (SCREEN_UPDATE) requires two parameters.
TPU$_TOOMANY	ERROR	You specified more than two parameters.
TPU$_UNKKEYWORD	ERROR	You have specified an unknown keyword.

STR

Accepts a string as a parameter without generating an error; also has a new optional parameter.

The STR built-in is supported in previous versions of VAXTPU. STR continues to perform all the functions it performed in previous versions; the *string1* parameter and the keywords ON and OFF are new in this version of VAXTPU.

For more information about the STR built-in, see the *VAX Text Processing Utility Manual*.

FORMAT

str3 := STR ({ { *buffer* / *range* } [[, *string2*]] [[, *ON* / , *OFF*]] / *string1* })

PARAMETERS

buffer

The buffer whose contents you want returned as a string.

range

The range whose contents you want returned as a string.

string1

Any string. STR now accepts a parameter of type STRING, so you need not check the type of the parameter you supply to the built-in. If you specify a parameter of type STRING, you cannot use either of the optional parameters.

string2

A string specifying how you want line ends represented. The default is the null string. You can only use *string2* if you specify a range or buffer as the first parameter. If you want to specify the keyword ON or OFF but do not want to specify *string2*, you must use a comma before the keyword as a placeholder, as follows:

```
new_string := STR (old_buffer, , ON);
```

ON

A keyword directing VAXTPU to insert spaces preserving the white space created by the left margin of each record in the specified buffer or range. Specifically, if you specify a buffer or range with a left margin greater than 1, the keyword ON directs VAXTPU to insert a corresponding number of spaces after the line-ends in the resulting string. For example, if the left margin of the specified lines is 10 and you use the keyword ON, VAXTPU inserts 9 spaces after each line-end in the resulting string. VAXTPU does not insert any spaces after line-begins of lines that do not contain characters. If the first line of a buffer or range starts at the left margin, VAXTPU inserts spaces before the text in the first line.

Note that you can only use this keyword if you specify a buffer or range as a parameter.

OFF

A keyword directing VAXTPU to ignore the left margin setting of the records in the specified buffer or range. This is the default. For example, if the left margin of the specified lines is 10 and you use the keyword OFF, VAXTPU does not insert any spaces after the line-ends in the resulting string.

Note that you can only use this keyword if you specify a buffer or range as a parameter.

return value

str3 The string equivalent of the parameter you specify.

SIGNALED ERRORS

TPU$_TRUNCATE	WARNING	You specified a buffer or range so large that converting it would exceed the maximum length for a string. VAXTPU has truncated characters from the returned string.
TPU$_NEEDTOASSIGN	ERROR	STR must appear on the right-hand side of an assignment statement.
TPU$_TOOFEW	ERROR	STR requires at least one argument.
TPU$_TOOMANY	ERROR	STR accepts only two arguments.
TPU$_INVPARAM	ERROR	The argument to STR must be an integer, buffer, string, or range.
TPU$_STRTOOLARGE	ERROR	The resulting string contains more than 65,535 characters.

EXAMPLES

1
```
return_string := STR (SELECT_RANGE, "<CRLF>", ON);
```

This statement creates a string using the text in the select range. Line breaks are marked with the string *CRLF*. The white space created by the margin is preserved by inserting spaces after the line breaks.

2
```
still_a_string := STR ("confetti");
```

This statement assigns the string *confetti* to the variable *still_a_string*.

5 VMS DECwindows VAXTPU Built-In Procedures

This chapter describes the new DECwindows-related built-in procedures in VAXTPU Version 2.2. For information about the new and modified built-ins that are common to both the DECwindows and non-DECwindows versions of VAXTPU, see Chapter 4. For information about the VAXTPU built-ins released before VMS Version 5.1, see the *VAX Text Processing Utility Manual*. For an overview of recent VAXTPU release history and version numbering, see Chapter 1.

5.1 Viewing Examples of Code Using Built-In Procedures

To see examples of how a layered application uses various VAXTPU built-ins, you can view some of the files used to create the EVE editor. To see a directory of the files available, type the following command from the DCL command line:

```
$ DIR SYS$EXAMPLES:EVE$*.TPU
```

These files contain procedures using almost all of the new and modified built-ins.

5.2 Descriptions of Built-In Procedures

This section describes the DECwindows VAXTPU built-in procedures that have been added in VAXTPU Version 2.2. The built-ins are listed alphabetically.

Using the new and modified built-ins, you can perform the following tasks:

- Create and manipulate widgets
- Enable, disable, and manipulate scroll bars
- Use the DECwindows clipboard with VAXTPU
- Use DECwindows global selection with VAXTPU
- Manipulate input focus
- Convert pixel-oriented information to character-cell-oriented information and the reverse
- Support mouse operations needed in a DECwindows environment
- Resize the VAXTPU screen
- Specify the application name that appears in the DECwindows icon box

For more information about using VAXTPU built-ins, see the *VAX Text Processing Utility Manual*.

CREATE_WIDGET

Creates a widget instance. The CREATE_WIDGET built-in has two variants with separate syntaxes. One variant creates and returns a widget using the intrinsics or an XUI Toolkit low-level creation routine. The other variant creates an entire hierarchy of widgets (as defined in an XUI Resource Manager database) and returns the topmost widget.

FORMAT

widget := CREATE_WIDGET
(widget_class, widget_name,
{ parent_widget }
{ SCREEN }
[, { , buffer }
{ , learn_sequence }
{ , program }
{ , range }
{ , string }]
[, closure]
[, widget_args]])

DESCRIPTION

Creates the widget instance you specify, using the intrinsics or an XUI Toolkit low-level creation routine. Although it has been created, the returned widget is not managed and therefore not visible. The application must call the MANAGE_WIDGET built-in to make the widget visible.

FORMAT

widget := CREATE_WIDGET
(resource_manager_name, hierarchy_id,
{ parent_widget }
{ SCREEN }
[, { , buffer }
{ , learn_sequence }
{ , program }
{ , range }
{ , string }]
[, closure]
[, widget_args]])

DESCRIPTION

Creates and returns an entire hierarchy of widgets (as defined in an XUI Resource Manager database) and returns the topmost widget. All children of the returned widget are also created and managed. The topmost widget is not managed, so none of the widgets created is visible.

If you specify one or more callback arguments in your User Interface Language (UIL) file, specify either the routine TPU$WIDGET_INTEGER_CALLBACK or the routine TPU$WIDGET_STRING_CALLBACK. For more information about specifying callbacks, see Chapter 6. For more information about UIL files, see the *VMS DECwindows Guide to Application Programming*.

When you use CREATE_WIDGET to create a widget or hierarchy of widgets organized by the XUI Resource Manager, CREATE_WIDGET uses the XUI Toolkit routine FETCH WIDGET.

PARAMETERS

widget_class

The integer returned by DEFINE_WIDGET_CLASS that specified the class of widget to be created.

widget_name

A string that is the name to be given to the widget.

parent_widget

The widget that is to be the parent of the newly created widget.

SCREEN

A keyword indicating that the newly created widget is to be the child of VAXTPU's main window widget.

buffer

The buffer containing the interface callback routine. This code is executed when the widget performs a callback to VAXTPU; all widgets created with a single CREATE_WIDGET call use the same callback code. If you do not specify this parameter, VAXTPU does not execute any callback code when the widget performs a callback to VAXTPU.

learn_sequence

The learn sequence that is the interface callback routine. This is executed when the widget performs a callback to VAXTPU; all widgets created with a single CREATE_WIDGET call use the same callback code. If you do not specify this parameter, VAXTPU does not execute any callback code when the widget performs a callback to VAXTPU.

program

The program that is the interface callback routine. This is executed when the widget performs a callback to VAXTPU; all widgets created with a single CREATE_WIDGET call use the same callback code. If you do not specify this parameter, VAXTPU does not execute any callback code when the widget performs a callback to VAXTPU.

range
The range containing the interface callback routine. This is executed when the widget performs a callback to VAXTPU; all widgets created with a single CREATE_WIDGET call use the same callback code. If you do not specify this parameter, VAXTPU does not execute any callback code when the widget performs a callback to VAXTPU.

string
The string containing the interface callback routine. This is executed when the widget performs a callback to VAXTPU; all widgets created with a single CREATE_WIDGET call use the same callback code. If you do not specify this parameter, VAXTPU does not execute any callback code when the widget performs a callback to VAXTPU.

closure
A string or integer. VAXTPU passes the value to the application when the widget performs a callback to VAXTPU. For more information about using closures, see Chapter 6. If you do not specify this parameter, VAXTPU passes the closure value (if any) given to the widget in the UIL file defining the widget. If you specify the closure value with CREATE_WIDGET instead of in the UIL file, all widgets created with the same CREATE_WIDGET call have the same closure value.

widget_args
A series of pairs of resource names and resource values. You can specify a pair in an array or as a pair of separate parameters. If you use an array, you index the array with a string that is the name of the resource you want to set. The array element contains the value you want to assign to that resource. If you use a pair of separate parameters, use the following format:

```
resource_name_string, resource_value
```

Arrays and string/value pairs may be used together. Each array index and its corresponding element value, or each string and its corresponding value, must be valid widget arguments for the class of widget you are creating.

resource_manager_name
A case-sensitive string that is the name assigned to the widget in the UIL file defining the widget.

hierarchy_id
The hierarchy identifier returned by the SET (DRM_HIERARCHY) built-in. This identifier is passed to the XUI Resource Manager, which uses the identifier to find the resource name in the database.

---

return value

The newly created widget instance.

DESCRIPTION

The case of a widget's name in the User Interface Definition (UID) file must match the case of the widget's name that you specify as a parameter to CREATE_WIDGET. If you specify case-sensitive widget names in your UIL file, you must use the same widget name case with CREATE_WIDGET as you used in the UIL file. If you specify case-insensitive widget names in your UIL file, the UIL compiler translates all widget names to

uppercase, so in this instance you must use uppercase widget names with CREATE_WIDGET. The example in the following section specifies case-insensitive widget names in the UIL file and specifies an uppercase name for the widget with the CREATE_WIDGET built-in.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_UNDWIDCLA	WARNING	You have specified a widget class integer that is not known to VAXTPU.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NEEDTOASSIGN	ERROR	The built-in must return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.
TPU$_WIDMISMATCH	ERROR	You have specified a widget whose class is not supported.

EXAMPLES

For an explanation of the code in these examples, see the paragraph following each numbered code example.

1

```
PROCEDURE eve_display_example

LOCAL   example_widget,                 ! Variable assigned to the created widget.
        example_widget_name,            ! The name of the widget assigned
                                        ! to this variable must be uppercase
                                        ! if you specified case-insensitive
                                        ! widget names in the UIL file.

        example_hierarchy;              ! XUI Resource Manager
                                        ! hierarchy for this example.

ON_ERROR
    [OTHERWISE]:                        ! Traps errors.
ENDON_ERROR;

! Set the widget hierarchy. The default file spec is "SYS$LIBRARY: .UID"

example_hierarchy := SET (DRM_HIERARCHY, "mynode$dua0:[smith]example");

! The VAXTPU CREATE_WIDGET built-in needs the name of the widget
! defined in the UIL file.

example_widget_name := "EXAMPLE_BOX";   ! The widget EXAMPLE_BOX is
                                        ! defined in the file EXAMPLE.UIL.

! Create the widget if it has not already been created.
```

```
IF GET_INFO (example_widget, "type") <> WIDGET
THEN
    example_widget := CREATE_WIDGET (example_widget_name, example_hierarchy,
                                     SCREEN, "eve$kt_callback_routine");

   ! EVE defines eve$kt_callback_routine to be EVE's callback routine.
   ! You do not need to define it again if you are extending EVE.

ENDIF;

! Map "example_widget" to the screen using MANAGE_WIDGET.

MANAGE_WIDGET (example_widget);

RETURN (TRUE);

ENDPROCEDURE;
```

This procedure, *eve_display_example*, creates a modal dialog box widget and maps the widget to the VAXTPU screen.

The procedure shows how to use the variant of CREATE_WIDGET that returns an entire widget hierarchy. To create a widget or widget hierarchy using this variant, you must have available the compiled form of a User Interface Language (UIL) file specifying the characteristics of the widgets you want to create. DIGITAL recommends that you use one or more UIL files and the corresponding variant of CREATE_WIDGET whenever possible, because UIL is more efficient and because UIL files make it easier to translate your application into other languages. For more information about compiling and using UIL files, see the *VMS DECwindows Guide to Application Programming*.

2
```
MODULE    example
VERSION = 'V00-000'

! This is a sample UIL file that creates a modal dialog box containing
! the message "Hello World."

NAMES = case_insensitive

PROCEDURE
        tpu$widget_integer_callback (integer);

VALUE
        example_callback        : 1;
        example_button_label    : 'Acknowledged';
        example_message         : 'Hello World';

OBJECT
        example_box : message_box {

            arguments {
                default_position = true;        ! puts box in center work area
                ok_label = example_button_label;
                label_label = example_message;
            };
        };

END MODULE;
```

This example shows a sample UIL file describing the modal dialog box called *example_box*. The UIL file specifies where the widget appears on the screen, what label appears on the box's button, and what message the widget displays.

For an example showing how to use the variant of CREATE_WIDGET that calls the XUI Toolkit low-level creation routine, see Example 7–2.

DEFINE_WIDGET_CLASS

Defines a widget class for later use in creating widgets of that class using the DECwindows intrinsics or the XUI Toolkit low-level creation routines.

FORMAT

integer := DEFINE_WIDGET_CLASS
(class_name
[[, creation_routine_name
[[, creation_routine_image_name]]])

PARAMETERS

class_name

A string that is the name of a universal symbol pointing to the desired widget class record. A universal symbol is a symbol in a shareable image that can be referred to in an image other than the one in which the symbol is defined.

creation_routine_name

A string that is the name of the low-level widget creation routine for this widget class. Specify the case of the string correctly. To determine the correct case of the string, consult the documentation for the widget whose class you are defining. The current version of VAXTPU, which is bundled with the VMS operating system, ignores the case of the string. However, future versions of VAXTPU may treat the string as case-sensitive.

If you do not specify this parameter, VAXTPU uses the XUI Toolkit CREATE WIDGET routine to create the widget instead of using a low-level widget creation routine. The routine must have the same calling sequence as the XUI Toolkit low-level widget creation routines.

creation_routine_image_name

A string that is the name of the shareable image in which the class record can be found. If you specify a low-level creation routine, DEFINE_WIDGET_CLASS also looks for the routine in the program image. If you do not specify an image, VAXTPU assumes the widget is defined in SYS$LIBRARY:DECW$DWTLIBSHR.EXE.

return value

An integer used to identify the class of widget to be created by the CREATE_WIDGET built-in.

DESCRIPTION

Each call returns a different class integer, which you use to specify the class of a widget when you create it.

SIGNALED ERRORS

TPU$_ARGMISMATCH	ERROR	The data type of the indicated parameter is not supported by the built-in.
TPU$_NEEDTOASSIGN	ERROR	The built-in must return a value.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.

EXAMPLE

For a sample procedure using the DEFINE_WIDGET_CLASS built-in, see Example 7–2.

DELETE (widget)

Deletes the specified widget instance and its children.

The DELETE built-in is supported in previous versions of VAXTPU. DELETE continues to perform all the functions it performed in previous versions; the use of DELETE to delete widgets is new in DECwindows VAXTPU.

For more information on the DELETE built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

DELETE *(widget)*

PARAMETERS

widget
The widget to be deleted.

DESCRIPTION

When you use the DELETE (widget) built-in, all variables and array elements that refer to the widget are set to UNSPECIFIED. If an array element is indexed by the deleted widget, the array element is deleted as well.

SIGNALED ERRORS

TPU$_TOOFEW	ERROR	DELETE requires one argument.
TPU$_TOOMANY	ERROR	DELETE accepts only one argument.
TPU$_BADDELETE	ERROR	You attempted to delete a constant.

EXAMPLE

The following code fragment creates a modal dialog box widget and later deletes it. For purposes of this example, the procedure *user_callback_dispatch_routine* is assumed to be a user-written procedure that handles widget callbacks. For a sample DECwindows User Interface Language (UIL) file to be used with VAXTPU code creating a modal dialog box widget, see the example in the description of the CREATE_WIDGET built-in.

```
PROCEDURE sample_create_and_delete

LOCAL example_widget,
      example_widget_name,
      example_hierarchy;
```

```
example_hierarchy := SET (DRM_HIERARCHY, "mynode$dua0:[smith]example.uid");
example_widget_name := "EXAMPLE_BOX";
example_widget := CREATE_WIDGET (example_widget_name,
                                 example_hierarchy, SCREEN,
                                 "user_callback_dispatch_routine");
!     .
!     .
!     .
DELETE (example_widget);
ENDPROCEDURE;
```

GET_CLIPBOARD

Reads STRING format data from the clipboard and returns a string containing this data.

FORMAT

string := GET_CLIPBOARD

return value

A string consisting of the data read from the clipboard. Line breaks are indicated by a line-feed character (ASCII (10)).

DESCRIPTION

DECwindows provides a clipboard that allows you to move data between applications. Applications can write to the clipboard to replace previous data, and can read from the clipboard to get a copy of existing data. The data in the clipboard may be in multiple formats, but all the information in the clipboard must be written at the same time.

VAXTPU provides no clipboard support for applications not written for DECwindows.

SIGNALED ERRORS

TPU$_NEEDTOASSIGN	ERROR	The built-in must return a value.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.
TPU$_CLIPBOARDFAIL	WARNING	The clipboard has not returned any data.
TPU$_CLIPBOARDLOCKED	WARNING	VAXTPU cannot read from the clipboard because some other application has locked it.
TPU$_CLIPBOARDNODATA	WARNING	There is no STRING format data in the clipboard.
TPU$_TRUNCATE	WARNING	Characters have been truncated because you tried to add text that would exceed the maximum line length.
TPU$_STRTOOLARGE	ERROR	The amount of data in the clipboard exceeds 65,535 characters.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.

EXAMPLE

The following statement reads what is currently in the clipboard and assigns it to *new_string*:

```
the_string := GET_CLIPBOARD;
```

For an example of a procedure using the GET_CLIPBOARD built-in, see Example 7–3.

GET_DEFAULT

Returns the value of an X resource from the X resources database.

FORMAT

{ string3 / integer } := GET_DEFAULT *(string1, string2)*

PARAMETERS

string1

The name of the resource whose value you want GET_DEFAULT to fetch. Note that resource names are case-sensitive.

string2

The class of the resource. Note that resource class names are case-sensitive.

return value

The string equivalent of the resource value or 0 if the specified resource is not defined. Note that, if necessary, the application must convert the string to the data type appropriate to the resource.

DESCRIPTION

GET_DEFAULT is useful for initializing a layered application that uses an X defaults file. You can use the built-in only in the DECwindows environment.

SIGNALED ERRORS

TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.
TPU$_NEEDTOASSIGN	ERROR	The built-in must return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.

EXAMPLE

If you want to create an extension of EVE that enables use of an X defaults file to choose a keypad setting, you can use a GET_DEFAULT statement in a module_init procedure. For more information about extending EVE using a module_init procedure and the EVE$BUILD tool, see the *VAX Text Processing Utility Manual*.

The following code fragment shows the portion of a module_init procedure directing VAXTPU to fetch the value of a resource from the X resources database.

```
PROCEDURE application_module_init

LOCAL
    keypad_name;

    :
    :
    :

keypad_name := GET_DEFAULT ("user.keypad", "User.Keypad");

EDIT (keypad_name, UPPER); ! Convert the returned string to uppercase.

IF keypad_name <> '0'
THEN

   CASE keypad_name

      "EDT"       :  eve_set_keypad_edt ();
      "NOEDT"     :  eve_set_keypad_noedt ();
      "WPS"       :  eve_set_keypad_wps ();
      "NOWPS"     :  eve_set_keypad_nowps ();
      "NUMERIC"   :  eve_set_keypad_numeric ();
      "VT100"     :  eve_set_keypad_vt100 ();
      [INRANGE, OUTRANGE]  :  eve_set_keypad_numeric; ! If user has
                                                      ! used invalid value,
                                                      ! set the keypad to
                                                      ! NUMERIC setting.

   ENDCASE;

ENDIF;

    :
    :
    :

ENDPROCEDURE;
```

To provide a value for the GET_DEFAULT statement to fetch, an X defaults file would contain an entry similar to the following:

```
User.Keypad : EDT
```

GET_GLOBAL_SELECT

Supplies information about a global selection.

FORMAT

{ unspecified | string | integer | array } := GET_GLOBAL_SELECT
({ PRIMARY | SECONDARY | selection_name } , selection_property_name)

PARAMETERS

PRIMARY
A keyword indicating the layered application is requesting information about a property of the primary global selection.

SECONDARY
A keyword indicating the layered application is requesting information about a property of the secondary global selection.

selection_name
A string identifying the global selection whose property is the subject of the layered application's information request. Specify the selection name as a string if the layered application needs information about a selection other than the primary or secondary global selection.

selection_property_name
A string specifying the property whose value the layered application is requesting.

return value

unspecified	A data type indicating that the information requested by the layered application was not available.
string	The value of the specified global selection property. The return value is of type STRING if the value of the specified global selection property is of type STRING.
integer	The value of the specified global selection property. The return value is of type INTEGER if the value of the specified global selection property is of type INTEGER.
array	An array passing information about a global selection whose contents describe information that is not of a data type supported by VAXTPU. For example, the array could return information about a pixmap, an icon, or a span.

VAXTPU does not use or alter the information in the array; the application layered on VAXTPU is responsible for determining how the information is used, if at all. Since the array is used to receive information from other DECwindows applications, all applications that exchange information whose data type is not supported by VAXTPU should adopt a convention on how the information is to be used.

The element *array {0}* contains a string naming the data type of the information being passed. For example, if the information being passed is a span, the element contains the string "*SPAN*". The element *array {1}* contains either the integer 8, indicating that the information is passed as a series of bytes, or the integer 32, indicating that the information is passed as a series of longwords. If *array {1}* contains the value 8, the element *array {2}* contains a string and there are no array elements after *array {2}*. The string does not name anything, but rather is a series of bytes of information. As mentioned, the meaning and use of the information is agreed upon by convention among the DECwindows applications. To interpret this string, the application can use the SUBSTR built-in to obtain substrings one at a time, and the ASCII built-in to convert the data to integer format if necessary. For more information about using these VAXTPU elements, see the *VAX Text Processing Utility Manual*.

If *array {1}* contains the value 32, the element *array {2}* contains an integer. In this case, the array can have any number of elements after *array {2}*. These elements are numbered sequentially beginning at *array {3}*. All the elements contain integers. Each integer represents a longword of data. To determine how many longwords are being passed, an application can determine the length of the array and subtract 2 to allow for elements *array {0}* and *array {1}*.

DESCRIPTION

If an owner for the global selection exists, and if the owner provides the information requested in a format that VAXTPU can recognize, GET_GLOBAL_SELECT returns the information.

SIGNALED ERRORS

TPU$_ARGMISMATCH	ERROR	Wrong type of data sent to the built-in.
TPU$_NEEDTOASSIGN	ERROR	The built-in must return a value.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.

TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.
TPU$_GBLSELOWNER	WARNING	VAXTPU owns the global selection.
TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVGBLSELDATA	WARNING	The global selection owner provided data that VAXTPU cannot process.
TPU$_NOGBLSELDATA	WARNING	The global selection owner has indicated that it cannot provide the information requested.
TPU$_NOGBLSELOWNER	WARNING	You have requested information about an unowned global selection.
TPU$_TIMEOUT	WARNING	The global selection owner did not respond before the timeout period expired.

EXAMPLE

The following statement fetches the text in the primary global selection and assigns it to the variable *string_to_paste*.

```
string_to_paste := GET_GLOBAL_SELECT (PRIMARY, "STRING");
```

For another example of how to use the GET_GLOBAL_SELECT built-in, see Example 7–4.

GET_INFO (buffer_variable, "read_routine")

Returns the global selection read routine.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

{ integer | program | learn_sequence } := GET_INFO ({ *buffer* | *SCREEN* }, *"read_routine", GLOBAL_SELECT)*

PARAMETERS

buffer

The buffer with which the global selection read routine is to be associated.

SCREEN

A keyword directing VAXTPU to return the application's default global selection read routine.

"read_routine"

A string indicating that the built-in is to return a read routine.

GLOBAL_SELECT

A keyword indicating that the built-in is to return the global selection read routine.

return value

integer	The value 0 if a global selection read routine has not been set for either the buffer or the application.
program	The program implementing the global selection read routine for either the buffer or the application.
learn_sequence	The learn sequence implementing the global selection read routine for either the buffer or the application.

DESCRIPTION

Returns the program or learn sequence that VAXTPU executes when it owns a global selection and another application has requested information about that selection. If the application has not specified a global selection read routine, 0 is returned.

The first parameter indicates whether the application is asking for the buffer-specific read routine or the application's default read routine.

GET_INFO (key_name, "key_modifiers")

Fetches information about the specified key name.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(key_name, "key_modifiers")*

PARAMETERS

key_name

A VAXTPU keyword that is a valid name for a key.

"key_modifiers"

A string directing VAXTPU to return a bit-encoded integer indicating what key modifier or modifiers have been used to create the VAXTPU key name specified by the parameter *key_name.*

return value

A bit-encoded integer indicating what key modifier or modifiers have been used to create the VAXTPU key name specified by the parameter *key_name.*

DESCRIPTION

Returns a bit-encoded integer indicating what key modifier or modifiers have been used to create the VAXTPU key name specified by the parameter *key_name.* For more information about the meaning and possible values of key modifiers, see the description of the KEY_NAME built-in in this chapter.

VAXTPU defines four constants to be used when referring to or testing the numerical value of key modifiers. The correspondence between key modifiers, defined constants, and bit-encoded integers is as follows:

Key Modifier Keyword	Corresponding Constant	Corresponding Bit-Encoded Integer
SHIFT_MODIFIED	TPU$K_SHIFT_MODIFIED	1
CTRL_MODIFIED	TPU$K_CTRL_MODIFIED	2
HELP_MODIFIED	TPU$K_HELP_MODIFIED	4
ALT_MODIFIED	TPU$K_ALT_MODIFIED	8

Note that the keyword SHIFT_KEY may have been used to create a VAXTPU key name. SHIFT_KEY is not a modifier, it is a prefix. The SHIFT key, also called the GOLD key by the EVE editor, is pressed and released before any other key is pressed. In DECwindows, modifying keys such as the CTRL key are pressed and held down while the modified key is pressed.

Note, too, that if more than one key modifier was used with the KEY_NAME built-in, the value of the returned integer is the value of the sum of the integer representations of the key modifiers. For example, if you create a key name using the modifiers HELP_MODIFIED and ALT_MODIFIED, the built-in GET_INFO (key_name, "key_modifiers") returns the integer 12.

EXAMPLE

The first statement in the following code creates a VAXTPU key name for the key sequence produced by pressing the CTRL key, the SHIFT key, and the 4 key on the keypad, all at once. The new key name is assigned to the variable *key_name*. The second statement fetches the integer equivalent of this combination of key modifiers. The third statement displays the integer 3 in the message buffer. The IF clause of the fourth statement shows how to test whether a key name was created using a modifier. (Note, however, that this statement would not detect whether a key name was created using the keyword SHIFT_KEY.) The THEN clause shows how to fetch the key modifier keyword or keywords used to create a key name. The final statement displays the string KEY_NAME (KP4, SHIFT_MODIFIED, ALT_MODIFIED) in the message buffer.

```
new_key := KEY_NAME (KP4, SHIFT_MODIFIED, CTRL_MODIFIED);
modifier_value := GET_INFO (new_key, "key_modifiers");
MESSAGE (STR (modifier_value));
IF GET_INFO (new_key, "key_modifiers")
THEN
    the_name := GET_INFO (new_key, "name")
ENDIF;
MESSAGE (STR (the_name));
```

GET_INFO (SCREEN, "active_area")

Fetches information about the active area (the area in which VAXTPU ignores movements of the pointer cursor).

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual*.

FORMAT

{ **array** / **integer** } **:= GET_INFO** *(SCREEN, "active_area")*

PARAMETERS

SCREEN

A keyword indicating that GET_INFO is to fetch information about the VAXTPU screen.

"active_area"

A string requesting GET_INFO to return information on the location and dimensions of the application's active area.

return value

array	An array containing information on the location and dimensions of the active area.
integer	The value 0, returned if there is no active area.

DESCRIPTION

GET_INFO (SCREEN, "active_area") returns an array containing information on the location and dimensions of the application's active area. The active area is the region in a window in which VAXTPU ignores movements of the pointer cursor for purposes of distinguishing clicks from drags. When you press down a mouse button, VAXTPU interprets the event as a click if the upstroke occurs in the active area with the downstroke. If the upstroke occurs outside the active area, VAXTPU interprets the event as a drag operation.

A VAXTPU layered application can have only one active area at a time, even if the application has more than one window visible on the screen. An active area is only valid if you are pressing a mouse button. The default active area occupies one character cell. By default, the active area is located on the character cell pointed to by the pointer cursor.

For information on mouse button clicks, which are related to the concept of an active area, see the *XUI Style Guide*.

GET_INFO (SCREEN, "active_area") returns five pieces of information about the active area in integer-indexed elements of the returned array. You need not use the CREATE_ARRAY built-in before using GET_INFO (SCREEN, "active_area"); VAXTPU assigns a properly structured array to the return variable you specify. The structure of the array is as follows:

Array Element	Contents
array {1}	The window containing the active area
array {2}	The column forming the leftmost edge of the active area.
array {3}	The row forming the top edge of the active area.
array{4}	The width of the active area, expressed in columns.
array{5}	The height of the active area, expressed in rows.

GET_INFO (SCREEN, "event")

If called from within a global selection grab or ungrab routine, identifies the global selection that was grabbed or lost. If called from within a routine that responds to requests for information about a global selection, returns the information an application needs to respond to the selection event.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual*.

For more information about grabbing and ungrabbing a global selection, see the *VMS DECwindows Guide to Application Programming*.

FORMAT

```
{ integer        }
{ PRIMARY        }
{ SECONDARY      } := GET_INFO
{ selection_name }
{ array          }
    (SCREEN, "event",
    GLOBAL_SELECT)
```

PARAMETERS

SCREEN

A keyword used to preserve compatibility with future versions of VAXTPU.

"event"

A string indicating that GET_INFO is to supply information about a DECwindows event.

GLOBAL_SELECT

A keyword indicating that GET_INFO is to supply information about a global selection.

return value

integer	The integer 0, indicating that the built-in is not responding to a grab, an ungrab, or a selection information request.
PRIMARY	A keyword indicating that the global selection is the primary global selection.
SECONDARY	A keyword indicating that the global selection is the secondary global selection.
selection_name	A string naming the global selection that is the subject of the information request. This string is returned only if the selection is not the primary or secondary global selection.

array	A two-element array that is returned if the built-in is used in a global selection read routine. The first of the array's two elements contains a keyword or string identifying which of the global selections is the subject of an information request. The second element contains a string naming the global selection property (such as text or a line number) that is the subject of an information request.

DESCRIPTION

If called from within a global selection grab or ungrab routine, GET_INFO (SCREEN, "event", GLOBAL_SELECT) identifies the global selection that was grabbed or lost. The built-in returns a keyword if the global selection was the primary or secondary selection. GET_INFO (SCREEN, "event", GLOBAL_SELECT) returns a string naming the global selection if the grab or ungrab involves a global selection other than the primary or secondary selection.

If called from within a routine that responds to requests for information about a global selection, GET_INFO (SCREEN, "event", GLOBAL_SELECT) returns an array. The array contains the information an application needs to respond to the selection event. The array contains the following information:

- array {1}—The keyword PRIMARY, the keyword SECONDARY, or a string. This element identifies which global selection was read.
- array {2}—A string. This element identifies the global selection property about which information has been requested.

SIGNALED ERRORS

TPU$_BUILTININV	WARNING	The built-in has been called from outside a global selection read, grab, or ungrab routine.

GET_INFO (SCREEN, "global_select")

Indicates whether VAXTPU currently owns the specified global selection.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(SCREEN, "global_select",* { *PRIMARY* | *SECONDARY* | *selection_name* } *)*

PARAMETERS

SCREEN
A keyword indicating that GET_INFO is to fetch information about the VAXTPU screen.

"global_select"
A string indicating that GET_INFO is to fetch information about a global selection.

PRIMARY
A keyword directing VAXTPU to get information on the primary global selection.

SECONDARY
A keyword directing VAXTPU to get information on the secondary global selection.

selection_name
A string identifying the global selection about which VAXTPU is to get information.

return value

Returns the integer 1 if VAXTPU currently owns the specified global selection; 0 if it does not.

GET_INFO (SCREEN, "grab_routine")

Returns the application's global selection or input focus grab routine.

FORMAT

```
{ integer }
{ program } := GET_INFO (SCREEN,
{ learn_seq }
                         "grab_routine",
                         { GLOBAL_SELECT }
                         { INPUT_FOCUS   } )
```

PARAMETERS

SCREEN

A keyword used to preserve compatibility with future versions of VAXTPU.

"grab_routine"

A string indicating that GET_INFO is to fetch the application's grab routine.

GLOBAL_SELECT

A keyword indicating that GET_INFO is to fetch the global selection grab routine.

INPUT_FOCUS

A keyword indicating that GET_INFO is to fetch the input focus grab routine.

return value

integer	The integer 0, indicating the requested grab routine is not present.
program	The program designated as the application's global selection or input focus grab routine.
learn_seq	The learn sequence designated as the application's global selection or input focus grab routine.

GET_INFO (SCREEN, "icon_name")

Returns the string used as the layered application's name in the DECwindows icon box.

FORMAT

string := GET_INFO *(SCREEN, "icon_name")*

PARAMETERS

SCREEN

A keyword indicating that GET_INFO is to fetch information about the VAXTPU screen.

"icon_name"

A string indicating that GET_INFO is to fetch the string used as the layered application's name in the DECwindows icon box.

return value

The string used as the layered application's name in the DECwindows icon box.

GET_INFO (SCREEN, "input_focus")

Indicates whether VAXTPU currently owns the input focus. Input focus is the ability to process user input from the keyboard.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual*.

FORMAT

integer := GET_INFO *(SCREEN, "input_focus")*

PARAMETERS

SCREEN

A keyword indicating that GET_INFO is to fetch information about the VAXTPU screen.

"input_focus"

A string indicating that GET_INFO is to fetch information about the input focus.

return value

The integer 1 if VAXTPU owns the input focus, otherwise 0.

GET_INFO (SCREEN, "length")

Returns the current length of the screen (in rows).

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(SCREEN, "length")*

PARAMETERS

SCREEN

A keyword indicating that GET_INFO is to fetch information about the VAXTPU screen.

"length"

A string indicating that GET_INFO is to fetch the current length of the VAXTPU screen.

return value

The current length of the VAXTPU screen, in rows.

GET_INFO (SCREEN, "new_length")

Returns the length (in rows) that the screen will have after the resize action routine has been executed.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(SCREEN, "new_length")*

PARAMETERS

SCREEN
A keyword indicating that GET_INFO is to fetch information about the VAXTPU screen.

"new_length"
A string indicating that GET_INFO is to fetch the length (in rows) that the screen will have after the resize action routine has been executed.

return value

The length (in rows) that the screen will have after the resize action routine has been executed.

DESCRIPTION

Resize action routines should use the length returned by GET_INFO (SCREEN, "new_length") to determine the length of their windows. If used outside of a resize action routine, this length is the same as the current length of the screen.

GET_INFO (SCREEN, "new_width")

Returns the width (in columns) the screen will have after the resize action routine has been executed.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual*.

FORMAT

integer := GET_INFO *(SCREEN, "new_width")*

PARAMETERS

SCREEN
A keyword indicating that GET_INFO is to fetch information about the VAXTPU screen.

"new_width"
A string indicating that GET_INFO is to fetch the width (in columns) that the screen will have after the resize action routine has been executed.

return value

The width (in columns) that the screen will have after the resize action routine has been executed.

DESCRIPTION

Resize action routines should use the length returned by GET_INFO (SCREEN, "new_width") to determine the width of their windows. If used outside of a resize action routine, this width is the same as the current width of the screen.

GET_INFO (SCREEN, "old_length")

Returns the length (in rows) of the screen before the most recent resize event.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(SCREEN, "old_length")*

PARAMETERS

SCREEN

A keyword indicating that GET_INFO is to fetch information about the VAXTPU screen.

"old_length"

A string indicating that GET_INFO is to fetch the length (in rows) that the screen had before the most recent resize event.

return value

The length (in rows) that the screen had before the most recent resize event.

DESCRIPTION

The *old_length* value is initially set to the length of the screen at startup. This value is reset after VAXTPU processes a resize event and before VAXTPU executes the resize action routine.

GET_INFO (SCREEN, "old_width")

Returns the width (in columns) of the screen before the most recent resize event.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual*.

FORMAT

integer := GET_INFO *(SCREEN, "old_width")*

PARAMETERS

SCREEN

A keyword indicating that GET_INFO is to fetch information about the VAXTPU screen.

"old_width"

A string indicating that GET_INFO is to fetch the width (in columns) that the screen had before the most recent resize event.

return value

The width (in columns) that the screen had before the most recent resize event.

DESCRIPTION

The *old_width* value is initially set to the width of the screen at startup. This value is reset after VAXTPU processes a resize event and before VAXTPU executes the resize action routine.

GET_INFO (SCREEN, "read_routine")

Returns the global selection read routine.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

```
{ integer   }
{ program   } := GET_INFO ( { buffer } ,
{ learn_sequence }        { SCREEN }
    "read_routine",
    GLOBAL_SELECT)
```

PARAMETERS

buffer
The buffer with which the global selection read routine is to be associated.

SCREEN
A keyword directing VAXTPU to return the application's default global selection read routine.

"read_routine"
A string indicating that the built-in is to return a read routine.

GLOBAL_SELECT
A keyword indicating that the built-in is to return the global selection read routine.

return value

integer	The value 0 if a global selection read routine has not been set for either the buffer or the application.
program	The program implementing the global selection read routine for either the buffer or the application.
learn_sequence	The learn sequence implementing the global selection read routine for either the buffer or the application.

DESCRIPTION

Returns the program or learn sequence that VAXTPU executes when it owns a global selection and another application has requested information about that selection. If the application has not specified a global selection read routine, 0 is returned.

The first parameter indicates whether the application is asking for the buffer-specific read routine or the application's default read routine.

GET_INFO (SCREEN, "screen_limits")

Returns an integer-indexed array specifying the minimum and maximum screen length and width.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

array := GET_INFO *SCREEN, "screen_limits"*

PARAMETERS

SCREEN

A keyword indicating that GET_INFO is to fetch information about the VAXTPU screen.

"screen_limits"

A string indicating that GET_INFO is to fetch information about the minimum and maximum screen length and width.

return value

array — An integer-indexed array using four elements to specify the minimum and maximum screen width and length. The array indices and their corresponding elements are as follows:

1. The minimum screen width, in columns. This value must be at least 0 and less than or equal to the maximum screen width. The default value is 0.
2. The minimum screen length, in lines. This value must be at least 0 and less than or equal to the maximum screen length.The default value is 0.
3. The maximum screen width, in columns. This value must be greater than or equal to the minimum screen width and less than or equal to 255. The default value is 255.
4. The maximum screen length, in lines. This value must be greater than or equal to the minimum screen length and less than or equal to 255. The default value is 255.

GET_INFO (SCREEN, "time")

Returns the amount of time after requesting global selection information that VAXTPU waits for a reply. When the time has expired, VAXTPU assumes the request will not be answered.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual*.

FORMAT

string := GET_INFO *(SCREEN, "time", GLOBAL_SELECT)*

PARAMETERS

SCREEN

A keyword indicating that GET_INFO is to fetch information about the VAXTPU screen.

"time"

A string indicating that GET_INFO is to fetch information about how long VAXTPU waits to receive information.

GLOBAL_SELECT

A keyword indicating that GET_INFO is to fetch the global selection ungrab routine.

return value

string	A string in VMS delta time format indicating how long VAXTPU waits.

GET_INFO (SCREEN, "ungrab_routine")

Returns the application's global selection or input focus ungrab routine.

FORMAT

```
{ int   }
{ prog  } := GET_INFO
{ learn }
      (SCREEN,
      "ungrab_routine",
      { GLOBAL_SELECT }
      { INPUT_FOCUS   })
```

PARAMETERS

SCREEN
A keyword used to preserve compatibility with future versions of VAXTPU.

"ungrab_routine"
A string indicating that GET_INFO is to fetch the application's ungrab routine.

GLOBAL_SELECT
A keyword indicating that GET_INFO is to fetch the global selection ungrab routine.

INPUT_FOCUS
A keyword indicating that GET_INFO is to fetch the input focus ungrab routine.

return value

int	The integer 0. This value indicates the requested ungrab routine is not present.
prog	The program designated as the application's global selection or input focus ungrab routine.
learn	The learn sequence designated as the application's global selection or input focus ungrab routine.

DESCRIPTION

Returns the program or learn sequence that VAXTPU executes when it loses ownership of a global selection or of the input focus.

GET_INFO (SYSTEM, "enable_resize")

Indicates whether resize operations are enabled.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(SYSTEM, "enable_resize")*

PARAMETERS

SYSTEM

A keyword indicating that GET_INFO is to fetch information about global settings in VAXTPU.

"enable_resize"

A string indicating that GET_INFO is to indicate whether resize operations are enabled.

return value

The value 1 if resize operations are enabled, otherwise 0. By default, resize operations are not enabled.

GET_INFO (SYSTEM, "resize_action")

Returns the current resize action routine.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

{ integer / program / learn_sequence } := GET_INFO *(SYSTEM, "resize_action")*

PARAMETERS

SYSTEM

A keyword indicating that GET_INFO is to fetch information about global settings in VAXTPU.

"resize_action"

A string indicating that GET_INFO is to fetch the application's resize action routine if it is present.

return value

integer	The integer 0, indicating the requested resize action routine is not present.
program	The program designated as the application's resize action routine.
learn_sequence	The learn sequence designated as the application's resize action routine.

GET_INFO (WIDGET, "callback_parameters")

Fetches the widget instance performing the callback, the closure value associated with the widget instance, and the reason for the callback.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

integer := GET_INFO *(WIDGET, "callback_parameters", array)*

PARAMETERS

WIDGET

A keyword directing GET_INFO to fetch information about VAXTPU widgets in general or about a specific widget whose name you do not know at the time you use the built-in.

"callback_parameters"

A string directing GET_INFO to get information about the widget that is calling back, the reason for the callback, and the closure associated with the widget.

array

An array with the following indices: "widget", "closure", and "reason_code". The built-in places the corresponding values in the array elements.

return value

integer	The value 0 if the call was encountered outside a callback procedure, 1 if the call was encountered within a callback procedure and callback information is available.

DESCRIPTION

VAXTPU automatically creates the array into which the return values are placed.

Use this built-in only in a widget callback procedure. If you use this built-in outside a widget callback procedure, the value returned is indeterminate.

For more information about callbacks and closure values in DECwindows VAXTPU, see Chapter 7. For general information about using callbacks and closure values, see the *VMS DECwindows Guide to Application Programming*.

EXAMPLE

The following procedure shows one possible way that a layered application can use GET_INFO (WIDGET, "callback_parameters", array). The procedure is a simplified version of the EVE procedure *eve$callback_dispatch*. You can find the original version in SYS$EXAMPLES:EVE$MENUS.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section 7.7.

This version of *eve$callback_dispatch* handles event callbacks from EVE widgets. The statement GET_INFO (WIDGET, "callback_parameters", temp_array) copies the following three items into elements of the array *temp_array*:

- The widget that is calling back
- The widget's integer closure
- The reason why the widget is calling back

The array *eve$$x_widget_array* contains pointers to all of EVE's callback routines in elements indexed by the appropriate integer closure values. This procedure locates the correct index in the array and executes the corresponding callback routine.

Warning: This simplified version of *eve$callback_dispatch* does not completely replace the version in existing EVE Version 2.2 code. Furthermore, DIGITAL does not guarantee that this example will work successfully with future versions of EVE. This example is presented solely to illustrate how EVE uses the built-in GET_INFO (WIDGET, "callback_parameters", array) in a callback handling procedure.

```
                    PROCEDURE eve$callback_dispatch

LOCAL   the_program,
        status,
        temp_array;

ON_ERROR
    [TPU$_CONTROLC]:
        eve$$x_state_array {eve$$k_command_line_flag} := eve$k_invoked_by_key;
        eve$learn_abort;
        ABORT;
    [OTHERWISE]:
        eve$$x_state_array {eve$$k_command_line_flag} := eve$k_invoked_by_key;
ENDON_ERROR

IF NOT eve$x_decwindows_active
THEN
    RETURN (FALSE);
ENDIF;

eve$$x_state_array {eve$$k_command_line_flag} := eve$k_invoked_by_menu;

status :=
     GET_INFO (WIDGET, "callback_parameters", temp_array); ! This statement using
                                                           ! GET_INFO (WIDGET)
                                                           ! returns the calling
                                                           ! widget, the reason
                                                           ! code, and the closure.
```

```
! The following statements make the contents of "temp_array"
! available to all the eve$$widget_xxx procedures

eve$x_widget := temp_array {"widget"};
                ! This array element contains the widget
                ! that called back.
eve$x_widget_tag := temp_array {"closure"};
                ! This array element contains the widget tag
                ! that is assigned to the widget in the UIL file.
eve$x_widget_reason := temp_array {"reason_code"};
                ! This array element contains callback reason code.

! The next line gets the callback routine from the array indexed
! by closure values.

the_program := eve$$x_widget_array {eve$x_widget_tag};

IF the_program <> 0
THEN
    EXECUTE (the_program);
ENDIF;

eve$$x_state_array {eve$$k_command_line_flag} := eve$k_invoked_by_key;
RETURN;

ENDPROCEDURE;
```

GET_INFO (WIDGET)

Fetches the widget instance corresponding to the specified widget name.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

widget := GET_INFO *(WIDGET, "widget_id",* { *parent_widget,* / *SCREEN,* } *widget_name* *)*

PARAMETERS

WIDGET

A keyword directing GET_INFO to fetch information about VAXTPU widgets in general or about a specific widget whose name you do not know at the time you use the built-in.

"widget_id"

A string directing GET_INFO to return the widget instance whose name matches the specified widget name.

parent_widget

The parent of the widget instance returned by the built-in.

SCREEN

A keyword indicating VAXTPU's main window widget is the ancestor of the widget instance that you want the built-in to return.

widget_name

A string that is the fully qualified name of the widget you want the built-in to return. To specify this parameter correctly, start the string with the name of the widget's parent. Use the same name you used to specify the *parent_widget* parameter. If you used the *SCREEN* parameter instead of the *parent_widget* parameter, start the string with the following widget name:

```
tpu$mainwindow
```

Next, specify the names of the ancestors, if any, that occur in the widget hierarchy between the parent and the widget itself. Start with the ancestor just below the parent and specify progressively more immediate ancestors. Finally, specify the name of the widget you want the built-in to return. Separate all widget names with periods.

The fully qualified widget name is case-sensitive.

return value

widget	The widget instance whose name matches the specified widget name.

DESCRIPTION

GET_INFO (WIDGET, "widget_id") calls the X Toolkit routine NAME TO WIDGET.

For more information on DECwindows concepts such as parent widgets, ancestor widgets, and the distinction between widget classes and widget instances, see the *VMS DECwindows Guide to Application Programming*.

EXAMPLE

The following statement assigns to the variable *the_text_widget* the widget instance named by the string *NEW_DIALOG.NEW_TEXT*. The widget instance is the child of the widget instance assigned to the variable *new_dialog*.

```
the_text_widget := GET_INFO (WIDGET, "widget_id", new_dialog,
                             "NEW_DIALOG.NEW_TEXT");
```

GET_INFO (widget_variable, "callback_routine")

Fetches the callback routine for the specified widget.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

{ **program** | **learn_sequence** } **:= GET_INFO**
(widget_variable, "callback_routine")

PARAMETERS

widget_variable

The widget instance whose callback routine you want to fetch.

"callback_routine"

A string directing GET_INFO to fetch the specified widget's callback routine.

return value

program	The program designated as the application's callback routine.
learn_sequence	The learn sequence designated as the application's callback routine.

DESCRIPTION

Returns the VAXTPU program or learn sequence that VAXTPU should execute when a widget callback occurs for the specified widget instance. For more information about callbacks, see Chapter 6.

EXAMPLE

The following statement executes the callback routine for the widget *eve$x_replace_dialog*. Note that this statement is valid only after the Replace dialog box has been used at least once, because EVE does not create any dialog box until you have invoked it.

```
EXECUTE (GET_INFO (eve$x_replace_dialog,
                   "callback_routine"));
```

GET_INFO (widget_variable, "name")

Returns the name of the specified widget instance.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

string := GET_INFO *(widget_variable, "name")*

PARAMETERS

widget_variable

The widget instance whose name you want to fetch.

"name"

A string directing GET_INFO to fetch the specified widget's name.

return value

string	The name of the specified widget instance.

EXAMPLE

The following procedure displays the name of the widget instance specified by the variable *eve$x_replace_dialog*. To confirm that the widget has been created as expected, the procedure also displays a message identifying the data type of the variable's contents. Note that the procedure is valid only after the *Replace* dialog box has been used at least once, because EVE does not create any dialog box until you have invoked it.

A statement containing the built-in GET_INFO (widget, "name") can be useful in code implementing a debugging command that evaluates VAXTPU statements, expressions, and variables.

```
PROCEDURE sample_return_name

LOCAL status;

status := GET_INFO (eve$x_replace_dialog,
                    "name");

MESSAGE ("The data type of status is: ");
MESSAGE (STR (GET_INFO(status, "type")));
MESSAGE ("The value of status is: ");
MESSAGE (STR (status));

ENDPROCEDURE;
```

GET_INFO (widget_variable, "widget_info")

Provides the current values for one or more resources of the specified widget.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

status := GET_INFO *(widget_variable, "widget_info",* { *array* | *arg_pair [, arg_pair . . .]* } *)*

PARAMETERS

widget_variable

The widget instance whose resource values you want to fetch.

"widget_info"

A string directing GET_INFO to fetch the widget's specified resource values.

array

An array specifying the resources whose values are to be fetched. Each array index must be a string naming a valid resource for the specified widget. Note that resource names are case sensitive. The corresponding array element contains the value of the resource. The array can contain any number of elements.

arg_pair

A string naming a valid resource for the widget followed by a variable to store the value of the resource. Use the following format:

```
resource_name_string, resource_value
```

You can fetch as many resources as you want by using multiple pairs of arguments.

return value

status	An integer representing the status with which the built-in completed execution. You need not use the return variable in a statement that includes this built-in.

DESCRIPTION

This built-in is functionally equivalent to the XUI Toolkit routine GET VALUES.

If you specify the name of a resource that the widget does not support, VAXTPU signals the error TPU$_ARGMISMATCH.

For more information about specifying resources, see Chapter 6.

EXAMPLE

The following procedure, *user_widget_replace_all*, shows one possible way that a layered application can use GET_INFO (widget, "widget_info"). The procedure is a modified version of the EVE procedure *eve$$widget_replace_all*. You can find the original version in SYS$EXAMPLES:EVE$MENUS.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section 7.7.

Procedure *user_widget_replace_all* determines what user message to display in response to the EVE command REPLACE. The procedure uses GET_INFO (widget, "widget_info") to fetch the value of the resource *dwt$c_nvalue*. This resource represents whether the *Replace All* toggle button appears unshaded, in which case the resource's value is 0, or appears solid, in which case the value is 1.

```
PROCEDURE user_widget_replace_all

CONSTANT
      user_k_widget_name := "REPLACE_DIALOG.REPLACE_ALL";

LOCAL the_value,
      parent_widget,
      replace_all_button;

parent_widget := eve$x_replace_dialog;

replace_all_button := GET_INFO (WIDGET, "widget_id",
                                parent_widget,
                                user_k_widget_name);

GET_INFO (replace_all_button,                ! This statement uses
          "widget_info", eve$dwt$c_nvalue,   ! GET_INFO (widget, "widget_info")
          the_value);                        ! to fetch the value of the
                                             ! dwt$c_nvalue resource.

IF the_value
THEN
    MESSGE ("All instances will be replaced.");
ELSE
    MESSGE ("Not all instances will be replaced.");
ENDIF;

ENDPROCEDURE;
```

GET_INFO (widget_variable, "text")

Fetches the string that is the value of the specified simple text widget. (The value of a text widget is the text entered into the text widget by the user in response to a prompt in a dialog box.)

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

string := GET_INFO *(widget_variable, "text")*

PARAMETERS

widget_variable
The widget instance whose text you want to fetch.

"text"
A string directing GET_INFO to fetch the specified widget's text value.

return value

string — The text value of the specified widget instance.

DESCRIPTION

This built-in is eqivalent to the XUI Toolkit routine *dwt$s_text_get_string.*

If the specified widget is not of class SText, VAXTPU signals the status TPU$_WIDMISMATCH.

EXAMPLE

The following code fragment creates an EVE file name dialog box widget and assigns the widget to the variable *eve$x_needfilename_dialog*. Next, the fragment assigns to the variable *the_value* a string prompting you for the name of a file to which the buffer's contents should be written. The fragment uses the built-in GET_INFO (WIDGET, "widget_id") to assign the dialog box's label widget to the variable *child_of_box*. Finally, the fragment assigns to the label widget's *evedwtc_nlabel* resource the string contained in *the_value*.

```
eve$x_needfilename_dialog := create_widget ("NEEDFILENAME_DIALOG",
                                            eve$k_widget_hierarchy,
                                            SCREEN,
                                            eve$kt_callback_routine);

the_value := message_text ("Type filename for writing buffer ", 0,
                           get_info (the_buffer, "name"));

child_of_box := get_info (WIDGET, "widget_id",
                          eve$x_needfilename_dialog,
                          "NEEDFILENAME_DIALOG.NEEDFILENAME_LABEL");

status := set (WIDGET, child_of_box, eve$dwt$c_nlabel, the_value);
```

GET_INFO (window_variable, "scroll_bar")

Fetches the specified scroll bar widget if it exists, otherwise returns 0.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

{ integer / widget } := GET_INFO *(window_variable, "scroll_bar", { HORIZONTAL / VERTICAL })*

PARAMETERS

window_variable

The window whose scroll bar you want VAXTPU to fetch.

"scroll_bar"

A string directing VAXTPU to fetch a scroll bar in a VAXTPU window.

HORIZONTAL

A keyword directing VAXTPU to fetch the window's horizontal scroll bar.

VERTICAL

A keyword directing VAXTPU to fetch the window's vertical scroll bar.

return value

integer	The value 0 if the window does not have the requested scroll bar.
widget	The widget instance implementing the vertical or horizontal scroll bar associated with a window.

EXAMPLE

The following statement returns the vertical scroll bar widget associated with the current window:

```
the_bar := GET_INFO (CURRENT_WINDOW, "scroll_bar", VERTICAL);
```

For another example of code using GET_INFO (window_variable, "scroll_bar") see Example 7–6.

GET_INFO (window_variable, "scroll_bar_auto_thumb")

Indicates whether automatic adjustment of the specified scroll bar slider is enabled.

For more information about the GET_INFO built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

{ integer / widget } := GET_INFO *(window_variable, "scroll_bar_auto_thumb", { HORIZONTAL / VERTICAL })*

PARAMETERS

window_variable
The window whose scroll bar slider is the subject of the information request.

"scroll_bar_auto_thumb"
A string directing GET_INFO to indicate whether automatic adjustment of the scroll bar slider is enabled.

HORIZONTAL
A keyword specifying the horizontal scroll bar.

VERTICAL
A keyword specifying the vertical scroll bar.

return value

integer — The value 1 if automatic adjustment is enabled, 0 if it is disabled.

EXAMPLE

The following statement returns an integer indicating whether automatic adjustment is enabled for the vertical scroll bar slider associated with the current window.

```
status := GET_INFO (CURRENT_WINDOW,
                    "scroll_bar_auto_thumb", VERTICAL);
```

For another example of code using GET_INFO (window_variable, "scroll_bar_auto_thumb", WINDOW) see Example 7–6.

KEY_NAME

Returns a VAXTPU keyword for a key or a combination of keys, or creates a keyword used as a key name by VAXTPU. In this version, KEY_NAME accepts new keywords for the optional parameter that specifies key modifiers (such as the CTRL key in the sequence CTRL/F12). The KEY_NAME built-in still performs all the functions that it performed in the version of VAXTPU released with VMS Version 5.0.

For more information about the KEY_NAME built-in, see the *VAX Text Processing Utility Manual.*

FORMAT

keyword2 := KEY_NAME

({ *integer* / *key_name* / *string* }
[[{ , *SHIFT_KEY* / , *SHIFT_MODIFIED* / , *ALT_MODIFIED* / , *CTRL_MODIFIED* / , *HELP_MODIFIED* } [[, . . .]]]
[[, *FUNCTION* / , *KEYPAD*]])

PARAMETERS

integer

An integer that is either the integer representation of a keyword for a key, or is a value between 0 and 255 that VAXTPU interprets as a value of a character in the DEC Multinational Character Set.

key_name

A keyword that is the VAXTPU name for a key.

string

A string that is the value of a key from the main keyboard.

SHIFT_KEY

A keyword specifying that the key name created includes one or more shift keys. The keyword SHIFT_KEY specifies the VAXTPU shift key, not the key on the keyboard marked "Shift." The shift key is also referred to as the GOLD key in EVE. (See the description of the SET (SHIFT_KEY) built-in in the *VAX Text Processing Utility Manual.*)

SHIFT_MODIFIED

A keyword specifying that the key name created by the built-in includes the key marked SHIFT on the keyboard. The keyword SHIFT_MODIFIED specifies the key that toggles between uppercase and lowercase; not the key known as the GOLD key.

SHIFT_MODIFIED only modifies function keys and keypad keys.

ALT_MODIFIED
A keyword specifying that the key name created by the built-in includes the ALT key. Note that on most DIGITAL keyboards the ALT key is labeled COMPOSE CHARACTER.

ALT_MODIFIED only modifies function keys and keypad keys.

CTRL_MODIFIED
A keyword specifying that the key name created by the built-in includes the CTRL key.

CTRL_MODIFIED only modifies function keys and keypad keys.

HELP_MODIFIED
A keyword specifying that the key name created by the built-in includes the HELP key.

HELP_MODIFIED only modifies function keys and keypad keys.

FUNCTION
A parameter that specifies that the resulting key name is to be that of a function key.

KEYPAD
A parameter that specifies that the resulting key name is to be that of a keypad key.

---

return value

A VAXTPU keyword to be used as the name of a key.

DESCRIPTION

Using the KEY_NAME built-in, you can create key names that are modified by more than one key. For example, it is possible to create a name for a key sequence consisting of the GOLD key, the CTRL key, and an alphanumeric or keypad key.

The built-in GET_INFO (key_name, "key_modifiers") returns a bit-encoded integer whose value represents the key modifier or combination of key modifiers used to create a given key name. For more information about interpreting the integer returned, see the description of GET_INFO (key_name, "key_modifiers").

The built-in GET_INFO (keyword, "name") has been extended to return a string including all the key modifier keywords used to create a key name. For more information about fetching the string equivalent of a key name, see the description of GET_INFO (keyword, "name").

SIGNALED ERRORS

TPU$_INCKWDCOM	WARNING	Inconsistent keyword combination.
TPU$_MUSTBEONE	WARNING	String must be one character long.

TPU$_NOTDEFINABLE	WARNING	Second argument is not a valid reference to a key.
TPU$_NEEDTOASSIGN	ERROR	Built-in call must be on the right-hand side of an assignment statement.
TPU$_ARGMISMATCH	ERROR	Wrong type of data sent to the built-in.
TPU$_BADKEY	ERROR	KEY_NAME accepts SHIFT_KEY, FUNCTION, or KEYPAD as a keyword argument.
TPU$_TOOFEW	ERROR	Too few arguments passed to the KEY_NAME built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the KEY_NAME built-in.

EXAMPLE

The following statements create a name for the key sequence GOLD/CTRL/KP4 and bind the EVE command FILL to the resulting key sequence:

```
new_key := KEY_NAME (KP4, CTRL_MODIFIED, SHIFT_KEY);
DEFINE_KEY ("eve_fill", new_key);
```

MANAGE_WIDGET

Makes the specified widget instance visible, provided that the specified widget's parent is also visible.

FORMAT

MANAGE_WIDGET *(widget [, widget . . .])*

PARAMETERS

widget
The widget instance to be managed.

DESCRIPTION

This built-in performs the same functions as the XUI Toolkit MANAGE CHILD and MANAGE CHILDREN routines.

If you have multiple children of a single widget that you want to manage, include them in a single call to MANAGE_WIDGET. Managing several widgets at once is more efficient than managing one widget at a time.

All widgets passed in the same MANAGE_WIDGET operation must have the same parent.

SIGNALED ERRORS

TPU$_INVPARAM	ERROR	You specified a parameter of the wrong type.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_WIDMISMATCH	ERROR	You have specified a widget whose class is not supported.

EXAMPLE

For a sample procedure using the MANAGE_WIDGET built-in, see Example 7–2.

READ_CLIPBOARD

Reads STRING format data from the clipboard and copies it into the current buffer, at the editing point, using the buffer's current text mode (INSERT or OVERSTRIKE).

FORMAT

[range | UNSPECIFIED] := READ_CLIPBOARD

return value

range	A range containing the text copied into the current buffer.
unspecified	A data type indicating that no data was obtained from the clipboard.

DESCRIPTION

If VAXTPU finds a line-feed character in the data, it removes the line feed and any adjacent carriage returns and puts the data after the line feed on the next line of the buffer. If VAXTPU must truncate the data from the clipboard, VAXTPU copies the truncated text into the current buffer.

All text read from the clipboard is copied into the buffer starting at the editing point. If VAXTPU must start a new line to fit all the text into the buffer, the new line starts at column 1, even if the current left margin is not set at column 1.

SIGNALED ERRORS

TPU$_CLIPBOARDLOCKED	WARNING	VAXTPU cannot read from the clipboard because some other application has locked it.
TPU$_CLIPBOARDNODATA	WARNING	There is no STRING format data in the clipboard.
TPU$_CLIPBOARDFAIL	WARNING	The clipboard has not returned any data.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_STRTOOLARGE	ERROR	The amount of data in the clipboard exceeds 65,535 characters.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following procedure shows one possible way that an application can use the READ_CLIPBOARD built-in. This procedure is a modified version of the EVE procedure *eve$$insert_clipboard*. You can find the original version in SYS$EXAMPLES:EVE$DECWINDOWS.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section 7.7.

Procedure *eve$$insert_clipboard* fetches the contents of the clipboard and places them in the current buffer.

```
PROCEDURE eve$$insert_clipboard

ON_ERROR
    [TPU$_CLIPBOARDNODATA]:
        eve$message (EVE$_NOINSUSESEL);
        eve$learn_abort;
        RETURN (FALSE);
    [TPU$_CLIPBOARDLOCKED]:
        eve$message (EVE$_CLIPBDREADLOCK);
        eve$learn_abort;
        RETURN (FALSE);
    [TPU$_TRUNCATE]:
    [OTHERWISE]:
        eve$learn_abort;
ENDON_ERROR;

IF eve$test_if_modifiable (CURRENT_BUFFER)
THEN
    READ_CLIPBOARD;                         ! This statement using
                                            ! READ_CLIPBOARD reads
                                            ! data from the clipboard
                                            ! and copies it into the
                                            ! current buffer.
    RETURN (TRUE);
ENDIF;

eve$learn_abort;
RETURN (FALSE);

ENDPROCEDURE;
```

READ_GLOBAL_SELECT

Requests information about the specified global selection from the owner of the global selection. If the owner provides the information, READ_GLOBAL_SELECT reads it and copies it into the current buffer at the editing point, using the buffer's current text mode (INSERT or OVERSTRIKE). The built-in also puts line breaks in the text copied into the buffer.

FORMAT

[{ unspecified / range } :=] READ_GLOBAL_SELECT
({ PRIMARY / SECONDARY / selection_name }, selection_property_name)

PARAMETERS

PRIMARY
A keyword indicating that the layered application is requesting information about a property of the primary global selection.

SECONDARY
A keyword indicating that the layered application is requesting information about a property of the secondary global selection.

selection_name
A string identifying the global selection whose property is the subject of the layered application's information request. Specify the selection name as a string if the layered application needs information about a selection other than the primary or secondary global selection.

selection_property_name
A string specifying the property whose value the layered application is requesting.

return value

unspecified	A data type indicating that the information requested by the layered application was not available.
range	A range containing the text copied into the current buffer.

DESCRIPTION

Use READ_GLOBAL_SELECT to ask the application that owns the specified global selection for information about a property of the global selection. For example, you can ask about the global selection's font, the number of lines it contains, or the string-formatted data it contains, if any.

All text read from the global selection is copied into the current buffer starting at the editing point. If VAXTPU must start a new line to fit all the text into the buffer, the new line starts at column 1, even if the current left margin is not set at column 1.

If the global selection information requested is an integer, the built-in converts the integer into a string before copying it into the current buffer. If the information requested is a string, the built-in copies the string into the buffer, replacing any line feeds with line breaks. Carriage returns adjacent to line feeds are not copied into the buffer.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_GBLSELOWNER	WARNING	VAXTPU owns the global selection.
TPU$_INVGBLSELDATA	WARNING	The global selection owner provided data that VAXTPU cannot process.
TPU$_NOGBLSELDATA	WARNING	The global selection owner has indicated that it cannot provide the information requested.
TPU$_NOGBLSELOWNER	WARNING	You have requested information about an unowned global selection.
TPU$_TIMEOUT	WARNING	The global selection owner did not respond before the timeout period expired.
TPU$_ARGMISMATCH	ERROR	Wrong type of data sent to the built-in.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following statement reads the string-formatted contents of the primary global selection and copies it into the current buffer at the current location.

```
READ_GLOBAL_SELECTION (PRIMARY, "STRING");
```

For another example of code using the READ_GLOBAL_SELECT built-in, see Example 7–9.

SET (ACTIVE_AREA)

Designates the specified area as the active area in a VAXTPU window. An active area is an area within which VAXTPU ignores movements of the pointer cursor.

FORMAT

SET *(ACTIVE_AREA, window, column, row [[, width, height]])*

PARAMETERS

ACTIVE_AREA
A keyword directing VAXTPU to set an attribute of the active area.

window
The window in which you want to define the active region.

column
An integer specifying the leftmost column of the active region.

row
An integer specifying the topmost row of the active region. If you use 0, the active row is the status line.

width
An integer specifying the width in columns of the active region. Defaults to 1.

height
An integer specifying the height in rows of the active region. Defaults to 1.

DESCRIPTION

The active area is the region in a window in which VAXTPU ignores movements of the pointer cursor for purposes of distinguishing clicks from drags. When you press down a mouse button, VAXTPU interprets the event as a click if the upstroke occurs in the active area with the downstroke. If the upstroke occurs outside the active area, VAXTPU interprets the event as a drag operation.

A VAXTPU layered application can have only one active area at a time, even if the application has more than one window visible on the screen. An active area is only valid if you are pressing a mouse button. The default active area occupies one character cell. By default, the active area is located on the character cell pointed to by the pointer cursor.

For information on mouse button clicks, which are related to the concept of an active area, see the *XUI Style Guide*.

Table 5–1 lists the keywords for referring to click and drag operations.

Table 5–1 VAXTPU Keywords Representing Mouse Events

M1UP	M2UP	M3UP	M4UP	M5UP
M1DOWN	M2DOWN	M3DOWN	M4DOWN	M5DOWN
M1DRAG	M2DRAG	M3DRAG	M4DRAG	M5DRAG
M1CLICK	M2CLICK	M3CLICK	M4CLICK	M5CLICK
M1CLICK2	M2CLICK2	M3CLICK2	M4CLICK2	M5CLICK2
M1CLICK3	M2CLICK3	M3CLICK3	M4CLICK3	M5CLICK3
M1CLICK4	M2CLICK4	M3CLICK4	M4CLICK4	M5CLICK4
M1CLICK5	M2CLICK5	M3CLICK5	M4CLICK5	M5CLICK5

SIGNALED ERRORS

TPU$_BADVALUE	ERROR	An integer parameter was specified with a value outside the valid range.
TPU$_EXTRANEOUSARGS	ERROR	One or more extraneous arguments have been specified for a DECwindows built-in.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following procedure shows one possible way that an application can use SET (ACTIVE_AREA). The procedure is a modified version of the EVE procedure *eve$$m1down*. You can find the original version in SYS$EXAMPLES:EVE$MOUSE.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section 7.7.

Procedure *eve$$m1down*, when bound to M1DOWN, sets an active area when you press MB1.

```
PROCEDURE eve$$m1down

LOCAL   the_window,
        the_column,
        the_row,
        the_width;
```

```
ON_ERROR
    [OTHERWISE]:
ENDON_ERROR;

eve$$x_pre_mb1_mark := MARK (FREE_CURSOR);

IF LOCATE_MOUSE (the_window, the_column, the_row)
THEN
    eve$x_mb1_in_progress := 1;
    IF the_row = 0
    THEN
        IF eve$current_indicator (the_window,
                                  the_column,
                                  the_width) <> 0
        THEN
            IF eve$x_decwindows_active
            THEN
                SET (ACTIVE_AREA,               ! This statement sets
                     the_window, the_column,    ! the active area.
                     0, the_width, 1);
            ENDIF;
        ELSE
            RETURN (FALSE);
        ENDIF;
    ELSE
        IF the_window = eve$choice_window
        THEN
            IF eve$$current_choice (the_column, eve$$x_chosen_range)
            THEN
                IF eve$x_decwindows_active
                THEN
                    SET (ACTIVE_AREA, the_window, the_column, the_row,
                         eve$$x_choices_column_width, 1);
                ENDIF;
            ENDIF;
        else
            POSITION (MOUSE);
            eve$$x_mb1_down_free := MARK (FREE_CURSOR);
            POSITION (TEXT);
            eve$clear_select_position;
            eve$clear_message;
            eve$$x_mb1_down_bound := MARK (NONE);
            POSITION (eve$$x_mb1_down_free);
        ENDIF;
    ENDIF;
    RETURN (TRUE);
ELSE
    RETURN (FALSE);
ENDIF;

ENDPROCEDURE;
```

SET (DRM_HIERARCHY)

Sets the User Interface Definition (UID) file or files to be used with VAXTPU.

FORMAT

integer := SET (DRM_HIERARCHY),
filespec
[[, filespec . . .]]

PARAMETERS

filespec
A string specifying the UID file to be used. VAXTPU applies the VMS default file specification "*SYS$LIBRARY:.UID*" to the string you pass to SET (DRM_HIERARCHY). You must specify at least one file name.

return value

An integer that is the identification number for the XUI Resource Manager hierarchy.

DESCRIPTION

Using UID files to specify hierarchies makes it easy to translate the product into other languages and to modify an application's interface without recompiling all the code implementing the application.

For more information about UID files, see the *VMS DECwindows Guide to Application Programming*.

SIGNALED ERRORS

TPU$_ARGMISMATCH	ERROR	The data type of the indicated parameter is not supported by the built-in.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following statement designates the User Interface Definition (UID) file MYNODE$DUA0:[SMITH]EXAMPLE.UID as a file to be used with VAXTPU to create widgets needed by the layered application:

```
example_hierarchy := SET (DRM_HIERARCHY, "mynode$dua0:[smith]example.uid");
```

For examples of how the SET (DRM_HIERARCHY) built-in is used in procedures, see Example 7–1 and Example 7–2.

SET (ENABLE_RESIZE)

Enables or disables resizing of the VAXTPU screen.

FORMAT

SET *(ENABLE_RESIZE,* { *ON* / *OFF* } *)*

PARAMETERS

ENABLE_RESIZE

A keyword directing VAXTPU to enable or disable screen resizing.

ON

A keyword enabling screen resizing.

OFF

A keyword disabling screen resizing.

DESCRIPTION

If you specify the ON keyword, VAXTPU gives the DECwindows window manager hints (parameters that the window manager is free to use or ignore) on the allowable maximum and minimum sizes for the VAXTPU screen. The hints are set by the SET (SCREEN_LIMITS, array) built-in. If you specify the OFF keyword, VAXTPU uses the screen's current width and length as the maximum and minimum size.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following statement enables screen resizing:

```
SET (ENABLE_RESIZE, ON);
```

To see this statement used in an initializing procedure, see the example in the description of the SET (SCREEN_LIMITS) built-in.

SET (GLOBAL_SELECT)

Requests ownership of the specified global selection property.

FORMAT

[[integer :=]] SET *(GLOBAL_SELECT, SCREEN,* { *PRIMARY* | *SECONDARY* | *selection_name* } *)*

PARAMETERS

GLOBAL_SELECT

A keyword indicating that GET_INFO is to fetch the global selection ungrab routine.

PRIMARY

A keyword directing VAXTPU to request ownership of the primary global selection.

SECONDARY

A keyword directing VAXTPU to request ownership of the secondary global selection.

selection_name

A string naming the global selection whose ownership VAXTPU is to request.

return value

The value 1 if the global selection ownership request was granted; 0 otherwise.

DESCRIPTION

The built-in returns the integer 1 if the request for ownership of a global selection was granted; otherwise 0.

The last parameter identifies the global selection of which VAXTPU is to grab ownership.

VAXTPU knows immediately if its request is granted. Therefore, VAXTPU does not automatically execute the global selection grab routine when it encounters this built-in. VAXTPU executes the routine only when it automatically grabs the primary selection.

For more information about the concept of global selection, see the *XUI Style Guide*.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following statement requests ownership of the primary global selection.

```
SET (GLOBAL_SELECT, SCREEN, PRIMARY);
```

For another example of code using the SET (GLOBAL_SELECT) built-in, see Example 7–10.

SET (GLOBAL_SELECT_GRAB)

Specifies the program or learn sequence VAXTPU should execute whenever it automatically grabs ownership of the primary selection.

FORMAT

SET *(GLOBAL_SELECT_GRAB, SCREEN*

[{ , buffer | , learn_sequence | , program | , range | , string | , NONE }])

PARAMETERS

buffer

The buffer that contains the grab routine. If you do not specify this parameter, the current global selection grab routine is deleted and your application is not informed when VAXTPU grabs the global primary global selection.

learn_sequence

The learn sequence specifying the grab routine. If you do not specify this parameter, the current global selection grab routine is deleted and your application is not informed when VAXTPU grabs the global primary global selection.

program

The program specifying the grab routine. If you do not specify this parameter, the current global selection grab routine is deleted and your application is not informed when VAXTPU grabs the global primary global selection.

range

The range that contains the grab routine. If you do not specify this parameter, the current global selection grab routine is deleted and your application is not informed when VAXTPU grabs the global primary global selection.

string

The string that contains the grab routine. If you do not specify this parameter, the current global selection grab routine is deleted and your application is not informed when VAXTPU grabs the global primary global selection.

NONE

A keyword directing VAXTPU to delete the current global selection grab routine.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

DESCRIPTION

For more information about VAXTPU's global selection support, see Section 7.2.

EXAMPLE

The following statement designates the procedure *user_grab_global* as a global selection read routine:

```
SET (GLOBAL_SELECT_GRAB, SCREEN, "user_grab_global");
```

For another example of code using the SET (GLOBAL_SELECT_GRAB) built-in, see Example 5–1.

Sample Code Setting Various Global Selection and Input Focus Routines

Example 5–1 shows possible ways that a layered application can use statements setting global selection and input focus routines. The example contains portions of the procedure *eve$mouse_module_init*. You can find the original version in SYS$EXAMPLES:EVE$MOUSE.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section 7.7.

The statements in Example 5–1 designate EVE's global selection read routine, global selection ungrab routine, global selection grab routine, input focus grab routine, and input focus ungrab routine.

Example 5–1 Initialization Procedure Using Variants of the SET Built-In

```
PROCEDURE eve$mouse_module_init

!  .
!  .
!  .

IF GET_INFO (SCREEN, "decwindows")
THEN

    SET (GLOBAL_SELECT_READ, SCREEN, "eve$write_global_select");
    SET (GLOBAL_SELECT_UNGRAB, SCREEN, "eve$global_select_ungrab");
    SET (GLOBAL_SELECT_GRAB, SCREEN, "eve$global_select_grab");
    SET (INPUT_FOCUS_GRAB, SCREEN, "eve$input_focus_grab");
    SET (INPUT_FOCUS_UNGRAB, SCREEN, "eve$input_focus_ungrab");
ENDIF;

ENDPROCEDURE;
```

SET (GLOBAL_SELECT_READ)

Specifies the program or learn sequence VAXTPU should execute whenever it receives a selection request event on a global selection it owns.

FORMAT

SET *(GLOBAL_SELECT_READ,* { *buffer1* / *SCREEN* } [{ *, buffer2* / *, learn_sequence* / *, program* / *, range* / *, string* / *, NONE* }] *)*

PARAMETERS

buffer1
The buffer with which the global selection read routine is to be associated.

SCREEN
A keyword indicating that the specified routine is to be the application's default global selection read routine.

buffer2
The buffer that contains the global selection read routine. If you do not specify this parameter, the global selection read routine is deleted.

learn_sequence
The learn sequence that specifies the global selection read routine. If you do not specify this parameter, the global selection read routine is deleted.

program
The program that specifies the global selection read routine. If you do not specify this parameter, the global selection read routine is deleted.

range
The range that contains the global selection read routine. If you do not specify this parameter, the global selection read routine is deleted.

string
The string that contains the global selection read routine. If you do not specify this parameter, the global selection read routine is deleted.

NONE
A keyword indicating that the global selection read routine should be deleted.

DESCRIPTION

To specify a buffer-specific global selection read routine, use the *buffer* parameter. To specify a global selection read routine for the entire application, use the SCREEN keyword.

When VAXTPU receives a request for information about a global selection it owns, it checks to see if the current buffer has a global selection read routine. If so, it executes that routine. If not, it checks to see if there is an application-wide global selection read routine. If so, it executes that routine. If not, it tries to respond to the request itself.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following statement designates the procedure *user_read_global* as a global selection read routine:

```
SET (GLOBAL_SELECT_READ, SCREEN, "user_read_global");
```

For another example of code using the SET (GLOBAL_SELECT_READ) built-in, see Example 5–1.

SET (GLOBAL_SELECT_TIME)

Specifies how long VAXTPU should wait before it assumes that a request for information about a global selection will not be satisfied.

FORMAT

SET *(GLOBAL_SELECT_TIME, SCREEN,* { *integer* | *string* } *)*

PARAMETERS

GLOBAL_SELECT_TIME

A keyword directing VAXTPU to set the expiration time for a global selection information request.

SCREEN

An optional keyword indicating that the top-level widget associated with VAXTPU's screen is to receive the input focus. This keyword is the default.

integer

The number of seconds that VAXTPU should wait.

string

A string in VMS delta time format indicating how long VAXTPU should wait.

DESCRIPTION

The default waiting time is set by DECwindows. The maximum waiting time you can set is 24 days, 20 hours.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVTIME	WARNING	You specified an invalid time interval.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following statement sets the waiting time for a global selection response to 3 seconds:

```
SET (GLOBAL_SELECT_TIME, SCREEN, 3);
```

SET (GLOBAL_SELECT_UNGRAB)

Specifies the program or learn sequence VAXTPU should execute whenever it loses ownership of a selection.

FORMAT

SET *(GLOBAL_SELECT_UNGRAB, SCREEN*

[{ , *buffer* | , *learn_sequence* | , *program* | , *range* | , *string* | , *NONE* }])

PARAMETERS

buffer

The buffer that contains the global selection ungrab routine. If you do not specify this parameter, the global selection ungrab routine is deleted and your application is not informed when VAXTPU gives up the global primary global selection.

learn_sequence

The learn sequence that specifies the global selection ungrab routine. If you do not specify this parameter, the global selection ungrab routine is deleted and your application is not informed when VAXTPU gives up the global primary global selection.

program

The program that specifies the global selection ungrab routine. If you do not specify this parameter, the global selection ungrab routine is deleted and your application is not informed when VAXTPU gives up the global primary global selection.

range

The range that contains the global selection ungrab routine. If you do not specify this parameter, the global selection ungrab routine is deleted and your application is not informed when VAXTPU gives up the global primary global selection.

string

The string that contains the global selection ungrab routine. If you do not specify this parameter, the global selection ungrab routine is deleted and your application is not informed when VAXTPU gives up the global primary global selection.

NONE

A keyword directing VAXTPU to delete the current global selection ungrab routine.

DESCRIPTION

For more information about VAXTPU's global selection support, see Section 7.2.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following statement designates the procedure *user_ungrab_global* as a global selection ungrab routine:

```
SET (GLOBAL_SELECT_UNGRAB, SCREEN, "user_ungrab_global");
```

For another example of code using the SET (GLOBAL_SELECT_UNGRAB) built-in, see Example 5–1.

SET (ICON_NAME)

Designates the string used as the layered application's name in the DECwindows icon box.

FORMAT

SET *(ICON_NAME, string)*

PARAMETERS

ICON_NAME

A keyword instructing VAXTPU to set the text of an icon.

string

The text you want to appear in the icon.

SIGNALED ERRORS

TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following statement sets the text naming the layered application to be the string *WordMonger*.

```
SET (ICON_NAME, "WordMonger");
```

SET (INPUT_FOCUS)

Requests the input focus. Ownership of the input focus determines which application or widget processes user input from the keyboard.

FORMAT

SET *(INPUT_FOCUS* [*, SCREEN* | *, widget*] *)*

PARAMETERS

INPUT_FOCUS

A keyword directing VAXTPU to assign the input focus.

SCREEN

An optional keyword indicating that the top-level widget associated with VAXTPU's screen is to receive the input focus. This keyword is the default.

widget

The widget that is to receive the input focus. Note that if you specify a widget for this parameter, the VAXTPU key bindings are not available to process keyboard input into the specified widget.

DESCRIPTION

This built-in requests that input focus be given to VAXTPU or to a widget that is part of an application layered on VAXTPU. It does not guarantee that VAXTPU or the widget will get the input focus. If VAXTPU or the widget receives the input focus, it gets a focus-in event. When VAXTPU gets this event, it calls the input focus grab routine. For more information about the role of events in DECwindows applications, see the *VMS DECwindows Guide to Application Programming*.

When the top-level widget for VAXTPU's screen has the input focus, VAXTPU processes keystrokes normally. That is, undefined printable keys insert characters in the current buffer, and defined keys execute the code bound to them. For more information on how VAXTPU processes keystrokes, see the *VAX Text Processing Utility Manual*.

If a child widget in the widget hierarchy has the input focus, keystrokes are processed by that widget. For example, when a text widget in EVE's replace dialog box has the input focus, keystrokes are processed by the text widget, not by VAXTPU. No VAXTPU key bindings are in effect.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.

TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following procedure shows one possible way that a layered application can use the SET (INPUT_FOCUS) built-in. The procedure is a modified version of the EVE procedure *eve$$widget_replace_okay*. You can find the original version in SYS$EXAMPLES:EVE$MENUS.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section 7.7.

Procedure *eve$$widget_replace_ok* fetches and tests the user's responses to prompts for old and new replace strings.

```
PROCEDURE eve$$widget_replace_ok

LOCAL   new_string,
        old_string,
        old_str_text_widget,
        new_str_text_widget;


SET (INPUT_FOCUS);      ! This statement grabs input focus
                        ! so CTRL/C events will be detected.

! Get the replace strings from the eve$$k_replace_new_[old]text widgets.

old_str_text_widget :=  GET_INFO (WIDGET, "widget_id", eve$x_replace_dialog,
                                  "REPLACE_DIALOG.REPLACE_OLD_TEXT")

old_string := GET_INFO (old_str_text_widget, "text");

! Test only the old string.
IF old_string = ""
THEN
    eve$message (EVE$_NOREPLSTR);
    RETURN;
ENDIF;

new_str_text_widget :=  GET_INFO (WIDGET, "widget_id", eve$x_replace_dialog,
                                  "REPLACE_DIALOG.REPLACE_NEW_TEXT")

new_string := GET_INFO (new_str_text_widget, "text");

IF new_string = ""
THEN
    eve$$replace1 (old_string, new_string, 1);
ELSE
    eve$$replace1 (old_string, new_string);
ENDIF;

ENDPROCEDURE;
```

SET (INPUT_FOCUS_GRAB)

Specifies the program or learn sequence that VAXTPU should execute whenever it processes a focus-in event.

FORMAT

SET *(INPUT_FOCUS_GRAB [[, SCREEN*

[[{ , buffer | , learn_sequence | , program | , range | , string | , NONE }]])

PARAMETERS

INPUT_FOCUS_GRAB

A keyword directing VAXTPU to set an attribute related to an input focus grab routine.

SCREEN

An optional keyword used to preserve compatibility with future versions of VAXTPU. This is the default.

buffer

The buffer that specifies the actions that VAXTPU should take whenever it processes a focus-in event.

learn_sequence

The learn sequence that specifies the actions that VAXTPU should take whenever it processes a focus-in event.

program

The program that specifies the actions that VAXTPU should take whenever it processes a focus-in event.

range

The range that specifies the actions that VAXTPU should take whenever it processes a focus-in event.

string

The string that specifies the actions that VAXTPU should take whenever it processes a focus-in event.

NONE

A keyword directing VAXTPU to delete the input focus grab routine. If you specify this keyword or do not specify the parameter at all, the application is not notified when input focus is received.

DESCRIPTION

For more information about VAXTPU's input focus support, see Section 7.3.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following statement designates the procedure *user_grab_focus* as an input focus grab routine:

```
SET (INPUT_FOCUS_GRAB, SCREEN, "user_grab_focus");
```

For another example of code using the SET (INPUT_FOCUS_GRAB) built-in, see Example 5–1.

SET (INPUT_FOCUS_UNGRAB)

Specifies the program or learn sequence that VAXTPU should execute whenever it processes a focus-out event.

FORMAT

SET *(INPUT_FOCUS_UNGRAB [[, SCREEN*

[[, buffer | , learn_sequence | , program | , range | , string | , NONE]])

PARAMETERS

INPUT_FOCUS_UNGRAB

A keyword directing VAXTPU to set an attribute related to an input focus ungrab routine.

SCREEN

An optional keyword used to preserve compatibility with future versions of VAXTPU. This is the default.

buffer

The buffer that specifies the actions that VAXTPU should take whenever it processes a focus-out event.

learn_sequence

The learn sequence that specifies the actions that VAXTPU should take whenever it processes a focus-out event.

program

The program that specifies the actions that VAXTPU should take whenever it processes a focus-out event.

range

The range that specifies the actions that VAXTPU should take whenever it processes a focus-out event.

string

The string that specifies the actions that VAXTPU should take whenever it processes a focus-out event.

NONE

A keyword directing VAXTPU to delete the input focus ungrab routine. If you specify this keyword or do not specify the parameter at all, the application is not notified when input focus is lost.

DESCRIPTION

For more information about VAXTPU's input focus support, see Section 7.3.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following statement designates the procedure *user_ungrab_focus* as an input focus ungrab routine:

```
SET (INPUT_FOCUS_UNGRAB, SCREEN, "user_ungrab_focus");
```

For another example of code using the SET (INPUT_FOCUS_UNGRAB) built-in, see Example 5–1.

SET (RESIZE_ACTION)

Specifies code to be executed when a resize event has occurred. Specifying a resize action routine overrides any previous resize action routines that have been defined.

FORMAT

SET *(RESIZE_ACTION* [, *buffer* | , *learn_sequence* | , *program* | , *range* | , *string* | , *NONE*] *)*

PARAMETERS

RESIZE_ACTION

A keyword directing VAXTPU to set an attribute related to a resize action routine.

buffer

The buffer that specifies the actions that VAXTPU should take whenever it is notified of a resize event.

learn_sequence

The learn sequence that specifies the actions that VAXTPU should take whenever it is notified of a resize event.

program

The program that specifies the actions that VAXTPU should take whenever it is notified of a resize event.

range

The range that specifies the actions that VAXTPU should take whenever it is notified of a resize event.

string

The string that specifies the actions that VAXTPU should take whenever it is notified of a resize event.

NONE

A keyword directing VAXTPU to delete the resize action routine. If you specify this keyword or do not specify the parameter at all, the application is not notified when a resize event occurs.

SIGNALED ERRORS

TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following statement specifies the procedure *eve$$resize_action* as the resize action routine.

```
SET (RESIZE_ACTION, "eve$$resize_action");
```

To see this statement used in an initializing procedure, see the example in the description of the SET (SCREEN_LIMITS) built-in.

SET (SCREEN_LIMITS)

Specifies the minimum and maximum allowable sizes for the VAXTPU screen during resize operations. VAXTPU passes these limits to the DECwindows window manager, which is free to use or ignore the limits.

FORMAT

SET *(SCREEN_LIMITS, array)*

PARAMETERS

SCREEN_LIMITS

A keyword directing VAXTPU to pass hints to the DECwindows window manager about screen size.

array

An integer-indexed array using four elements to specify hints for the minimum and maximum screen width and length. The array indices and their corresponding elements are as follows:

1 The minimum screen width, in columns. This value must be at least 0 and less than or equal to the maximum screen width. The default value is 0.

2 The minimum screen length, in lines. This value must be at least 0 and less than or equal to the maximum screen length.The default value is 0.

3 The maximum screen width, in columns. This value must be greater than or equal to the minimum screen width and less than or equal to 255. The default value is 255.

4 The maximum screen length, in lines. This value must be greater than or equal to the minimum screen length and less than or equal to 255. The default value is 255.

SIGNALED ERRORS

TPU$_BADVALUE	WARNING	An integer parameter was specified with a value outside the valid range.
TPU$_MAXVALUE	WARNING	You specified a value higher than the maximum allowable value.
TPU$_MINVALUE	WARNING	You specified a value lower than the minimum allowable value.
TPU$_EXTRANEOUSARGS	ERROR	One or more extraneous arguments has been specified for a DECwindows built-in.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.

TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.
TPU$_REQARGSMISSING	ERROR	One or more required arguments is missing.

EXAMPLE

The following statements show one possible way that a layered application can use the SET (SCREEN_LIMITS) built-in. The statements are a portion of the EVE procedure *eve$$decwindows_init*. You can find the original version in SYS$EXAMPLES:EVE$DECWINDOWS.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section 7.7.

The procedure *eve$$decwindows_init* is the module initialization procedure for the package EVE$DECWINDOWS.

```
PROCEDURE eve$$decwindows_init     ! Module Initialization

LOCAL   temp_array;

eve$x_decwindows_active := GET_INFO (SCREEN, "decwindows");

!  .
!  .
!  .

IF NOT eve$x_decwindows_active
THEN
    RETURN (FALSE)
ENDIF;

! The following statements set the package up to handle resize actions.

temp_array := CREATE_ARRAY (4);
temp_array {1} := 20;    ! Minimum width.
temp_array {2} := 6;     ! Minimum height.
temp_array {3} := 250;   ! Maximum width.
temp_array {4} := 100;   ! Maximum height.

SET (SCREEN_LIMITS, temp_array);
SET (RESIZE_ACTION, "eve$$resize_action");
SET (ENABLE_RESIZE, ON);

!  .
!  .
!  .

ENDPROCEDURE;
```

SET (SCROLL_BAR)

Enables a horizontal or vertical scroll bar for the specified window.

FORMAT

[{ integer / widget } :=] SET *(SCROLL_BAR, window, { HORIZONTAL, / VERTICAL, } { ON / OFF })*

PARAMETERS

SCROLL_BAR
A keyword directing VAXTPU to enable or disable a scroll bar in a VAXTPU window.

window
The window in which the scroll bar does or does not appear.

HORIZONTAL
A keyword directing VAXTPU to enable or disable a horizontal scroll bar.

VERTICAL
A keyword directing VAXTPU to enable or disable a vertical scroll bar.

ON
A keyword indicating that the scroll bar is to be visible in the specified window.

OFF
A keyword indicating that the scroll bar is not to be visible in the specified window.

return value

integer	The value 0 if an error prevents VAXTPU from associating a widget with the window.
widget	The widget instance implementing the vertical or horizontal scroll bar associated with a window.

DESCRIPTION

Scroll bars represent the location of the editing point in the buffer. By dragging the scroll bar's slider, the user can reposition the editing point in the buffer mapped to the window. Scroll bars are unique among VAXTPU widgets in the following respects:

- Each scroll bar widget is associated with a specific VAXTPU window.

- Instead of handling scroll widgets at the application level, you can direct the engine (VAXTPU's internal routines, taken as a whole) to handle resizing and repositioning of the scroll bar slider. The engine always handles sizing and positioning of the scroll bar itself.

Note that windows having fewer than four lines of text cannot display a vertical scroll bar. Similarly, a window less than four columns wide cannot display a horizontal scroll bar.

SET (SCROLL_BAR) returns the scroll bar widget, or 0 if an error prevents VAXTPU from associating a widget with the window.

By default, VAXTPU creates its windows without any scroll bars; using SET (SCROLL_BAR) with the keyword ON overrides the default. To make a scroll bar invisible after it has been placed in a window (for example, to allow the user of a layered application to turn off scroll bars), use SET (SCROLL_BAR) with the keyword OFF.

When the size of a VAXTPU window changes, VAXTPU automatically adjusts the scroll bar to fit the new window size.

The height of a vertical scroll bar represents the total number of lines in the buffer mapped to the window.

The width of a horizontal scroll bar represents the greater of the following:

- The width of the widest line in the set of lines visible in the window. "Width" means the distance from the first character on the line to the last character, regardless of whether all characters on the line are visible.
- In a case where none of the lines in the set of lines visible in the window has text extending all the way to the rightmost window column, the width of the widest line from the first character on the line to the rightmost window column.

Note that the horizontal scroll bar represents only the lines that are visible in the window, not all the lines in the buffer mapped to the window.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following statement turns on a vertical scroll bar in the current window:

```
vertical_bar := SET (SCROLL_BAR, CURRENT_WINDOW, VERTICAL, ON);
```

For sample code using the SET (SCROLL_BAR) built-in, see Example 7–7.

SET (SCROLL_BAR_AUTO_THUMB)

Enables or disables automatic adjustment of the scroll bar slider.

FORMAT

SET *(SCROLL_BAR_AUTO_THUMB, window, { HORIZONTAL / VERTICAL }, { ON / OFF })*

PARAMETERS

SCROLL_BAR_AUTO_THUMB

A keyword directing VAXTPU to enable or disable automatic adjustment of the scroll bar slider in a VAXTPU window.

window

The window whose scroll bar slider you want VAXTPU to adjust.

HORIZONTAL

A keyword directing VAXTPU to set the slider on a horizontal scroll bar.

VERTICAL

A keyword directing VAXTPU to set the slider on a vertical scroll bar.

ON

A keyword directing VAXTPU to enable automatic adjustment of the scroll bar slider.

OFF

A keyword directing VAXTPU to disable automatic adjustment of the scroll bar slider.

DESCRIPTION

By default, SET (SCROLL_BAR_AUTO_THUMB) is set to ON and VAXTPU automatically manages a window's scroll bar slider in the following ways:

- Adjusts the size of the slider as the user adds, deletes, or moves text, so that the slider size represents the amount of visible text in relation to the total amount of text
- Adjusts the size of the slider whenever the size of the window and the size of the scroll bar change, so that the slider size remains proportional to the scroll bar size
- Adjusts the position of the slider as the user adds, deletes, or moves text, so that the slider shows whether the current buffer or line contains text not visible on the screen and, if so, where the invisible text is in relation to the visible text

When the scroll bar slider is adjusted automatically, the width of the slider in a horizontal scroll bar represents the width of the window. For example, the size of the slider changes when the window width is changed from 80 to 132 columns or the reverse. The position of the slider changes when the window is shifted left or right. The height of the slider in a vertical scroll bar represents the height of the window.

If you do not want VAXTPU to adjust the scroll bar slider automatically or if you want to change the size or position of the slider, specify the OFF keyword. For more information about calculating the size and position of the slider, see the description of the SET (SCROLL_BAR) built-in.

Note that you cannot disable VAXTPU's automatic adjustment of the scroll bar itself.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following statement turns on automatic adjustment of the vertical scroll bar's slider in the current window:

```
vertical_bar := SET (SCROLL_BAR_AUTO_THUMB, CURRENT_WINDOW, VERTICAL, ON);
```

For sample code using the SET (SCROLL_BAR_AUTO_THUMB) built-in, see Example 7–7.

SET (TEXT)

Sets the text of the specified widget of class SText to be the specified string.

For more information about widget classes, see the *VMS DECwindows Toolkit Routines Reference Manual.*

FORMAT

SET *(TEXT, widget, string)*

PARAMETERS

TEXT

A keyword directing VAXTPU to set an attribute related to text manipulation.

widget

The widget instance whose text you want to set.

string

The text you want to assign to the simple text widget.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.
TPU$_WIDMISMATCH	ERROR	The specified widget is not of class SText.

DESCRIPTION

SET (TEXT, widget, string) is equivalent to the XUI Toolkit routine *dwt$s_text_set_string*.

EXAMPLES

1

```
SET (TEXT, user_text_widget, "No default string available.");
```

Assuming that the variable *user_text_widget* has been assigned a text widget instance, this statement causes the widget to display the text *No default string available.*

```
2   wildcard_dialog_box := GET_INFO (WIDGET, "widget_id",
                                     eve$x_wildcard_find_dialog,
                                     "WILDCARD_FIND_DIALOG.WILDCARD_FIND_TEXT");

    status := SET (TEXT, wildcard_dialog_box, eve$x_target);
```

These statements show one possible way that a layered application can use the SET (TEXT) widget. The variable *eve$x_target* stores the string (if one exists) that the user specified as the wildcard search string the last time the user invoked the wildcard find dialog box. The SET (TEXT) statement directs EVE's wildcard find dialog box widget to display the string assigned to *eve$x_target*.

SET (WIDGET)

Allows you to assign values to various resources of a widget.

FORMAT

SET *(WIDGET, widget,*
{ array
{ name_and_value [, name_and_value . . .] })

PARAMETERS

WIDGET

A keyword directing VAXTPU to set an attribute of a widget.

widget

The widget instance whose values you want to set.

array

An array specifying the resources whose values are to be set. Each array index must be a string naming a valid resource for the specified widget. Note that resource names are case-sensitive. The value in the corresponding array element is passed to the widget. The array can contain any number of elements.

name_and_value

A string naming a valid resource for the widget followed by the value that you want to assign to the resource. Use the following format:

```
resource_name_string, resource_value
```

DESCRIPTION

This built-in is functionally equivalent to the XUI Toolkit routine SET VALUES.

If you specify the name of a resource that the widget does not support, VAXTPU signals the error TPU$_ARGMISMATCH.

For more information about specifying resources, see Chapter 6.

SIGNALED ERRORS

TPU$_BADKEY	WARNING	You specified an invalid keyword as a parameter.
TPU$_ARGMISMATCH	ERROR	You have specified an unsupported class of widget or a value whose data type is not supported.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.

TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.
TPU$_WIDMISMATCH	ERROR	You have specified a widget whose class is not supported.

EXAMPLE

The following statements set the *Nvalue* resource of the current window's scroll bar widget to 100. This causes the scroll bar slider to be displayed as far toward the bottom of the scroll bar widget as possible.

```
scroll_bar_widget := SET (SCROLL_BAR, CURRENT_WINDOW, VERTICAL, ON);
SET (WIDGET, scroll_bar_widget, eve$dwt$c_Nvalue, 100);
```

For an example of a procedure using the SET (WIDGET) built-in, see Example 7–8.

SET (WIDGET_CALLBACK)

Specifies the VAXTPU program or learn sequence to be called by VAXTPU when a widget callback occurs for the widget instance.

FORMAT

SET *(WIDGET_CALLBACK, widget,*

{ *buffer,*
{ *learn_sequence,*
{ *program,* } *closure)*
{ *range,*
{ *string,*

PARAMETERS

WIDGET_CALLBACK
A keyword directing VAXTPU to set the application-level widget callback.

widget
The widget instance whose callback you want to set.

buffer
The buffer that contains the application-level callback routine. This code is executed when the widget performs a callback to VAXTPU.

learn_sequence
The learn sequence that specifies the application-level callback routine. This code is executed when the widget performs a callback to VAXTPU.

program
The program that specifies the application-level callback routine. This code is executed when the widget performs a callback to VAXTPU.

range
The range that contains the application-level callback routine. This code is executed when the widget performs a callback to VAXTPU.

string
The string that contains the application-level callback routine. This code is executed when the widget performs a callback to VAXTPU.

closure
A string or integer. VAXTPU passes the value to the application when the widget performs a callback to VAXTPU. For more information about using closures, see Chapter 6.

SIGNALED ERRORS

TPU$_ARGMISMATCH	ERROR	The data type of the indicated parameter is not supported by the built-in.
TPU$_BADDELETE	ERROR	You are attempting to modify an integer, a keyword, or a string constant.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.
TPU$_COMPILEFAIL	WARNING	Program compilation has been terminated because of a syntax error.

EXAMPLE

The following statement designates the EVE procedure *eve$scroll_dispatch* as the callback routine for the widget *scroll_bar_widget* and assigns to the callback the closure value *'h'*.

```
SET (WIDGET_CALLBACK, scroll_bar_widget, "eve$scroll_dispatch", 'h');
```

For a procedure using this statement while mapping windows see Example 7–7.

UNMANAGE_WIDGET

Makes the specified widget and all of its children invisible.

For more information about managing widgets, see the *VMS DECwindows Toolkit Routines Reference Manual.*

FORMAT

UNMANAGE_WIDGET *(widget [, widget . . .])*

PARAMETERS

widget
The widget instance to be unmanaged.

DESCRIPTION

If you want to unmanage several widgets that are children of the same parent, but you do not want to unmanage the parent, include all the children in a single call to UNMANAGE_WIDGET. Unmanaging several widgets at once is more efficient than unmanaging one widget at a time.

The UNMANAGE_WIDGET built-in is equivalent to the XUI Toolkit UNMANAGE CHILD and UNMANAGE CHILDREN routines.

SIGNALED ERRORS

TPU$_INVPARAM	ERROR	You specified a parameter of the wrong type.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_WIDMISMATCH	ERROR	You have specified a widget whose class is not supported.

EXAMPLE

The following procedure shows one possible way that a layered application can use the UNMANAGE_WIDGET built-in. The procedure is a modified version of the EVE procedure *eve$$replace_clean_up*. You can find the original version in SYS$EXAMPLES:EVE$EDIT.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section 7.7.

The procedure performs screen cleanup operations after the user has used the EVE command REPLACE. It restores the direction and mode to which the buffer was set before the replace operation began, then tests whether the replace dialog box is present and, if so, makes it invisible.

```
PROCEDURE eve$$replace_clean_up

ON_ERROR
    [TPU$_CONTROLC]:
        eve$learn_abort;
        abort;
    [OTHERWISE]:
        eve$$replace_error_handler;
ENDON_ERROR;

IF NOT eve$$x_replace_array {eve$$k_replace_asking}

THEN        ! If all occurrences were replaced, the editing
            ! point is positioned to the last saved_mark.

    POSITION (eve$$x_replace_array {eve$$k_replace_saved_mark});
ENDIF;

! Restore the buffer's original direction and mode.

SET (eve$$x_replace_array {eve$$k_replace_saved_direction},
     eve$$x_replace_array {eve$$k_replace_this_buffer});
SET (eve$$x_replace_array {eve$$k_replace_saved_mode},
     eve$$x_replace_array {eve$$k_replace_this_buffer});

SET (SCREEN_UPDATE, ON);
eve$message (EVE$_REPLCOUNT, 0,
             eve$$x_replace_array {eve$$k_replace_occurrences});

 IF (eve$$x_state_array {eve$$k_command_line_flag} = eve$k_invoked_by_menu)
    AND (eve$$x_state_array {eve$$k_dialog_box})
 THEN
    IF eve$x_decwindows_active
    THEN
        IF GET_INFO (eve$x_replace_each_dialog, "type") = WIDGET
        THEN

            UNMANAGE_WIDGET (eve$x_replace_each_dialog);    ! This statement
                                                            ! unmanages the
                                                            ! replace dialog
                                                            ! box.

        ENDIF;
    ENDIF;
 ENDIF;

ENDPROCEDURE;
```

WRITE_CLIPBOARD

Writes STRING format data to the clipboard.

FORMAT

WRITE_CLIPBOARD *(clipboard_label,* { *buffer* | *range* | *string* } *)*

PARAMETERS

clipboard_label

The label for multiple entries in the clipboard. Since the clipboard does not currently support multiple labels, use any string including the null string to specify this parameter.

buffer

The buffer containing text to be written to the clipboard. VAXTPU represents line breaks by a line-feed character (ASCII (10)). If you specify a buffer, VAXTPU converts the buffer to a string, replacing line breaks with line feeds, and replacing the white space before the left margin with padding blanks.

The buffer must contain at least one character or line break. If it does not, VAXTPU signals TPU$_CLIPBOARDZERO.

range

The range containing text to be written to the clipboard. VAXTPU represents line breaks by a line-feed character (ASCII (10)). If you specify a range, VAXTPU converts the range to a string, replacing line breaks with line feeds, and replacing the white space before the left margin with padding blanks.

The range must contain at least one character or line break. If it does not, VAXTPU signals TPU$_CLIPBOARDZERO.

string

The string containing text to be written to the clipboard. The string must contain at least one character. If it does not, VAXTPU signals TPU$_ CLIPBOARDZERO.

DESCRIPTION

The *clipboard_label* parameter provides support for multiple entries on the clipboard; however, at present, the clipboard does not support multiple entries.

SIGNALED ERRORS

TPU$_CLIPBOARDLOCKED	WARNING	The clipboard is locked by another process.

TPU$_CLIPBOARDZERO	WARNING	The data to be written to the clipboard has zero length.
TPU$_TRUNCATE	WARNING	VAXTPU has truncated characters from the data written because you specified a buffer or range containing more than 65,535 characters.
TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLES

For an explanation of the VAXTPU code in these examples, see the paragraph following each numbered code example.

1
```
WRITE_CLIPBOARD ("", this_range);
```

This statement writes the contents of the range *this_range* to the clipboard.

2
```
PROCEDURE eve$$cut_copy (delete_range)
LOCAL   remove_range,           ! Local copy of the currently
                                ! selected range.
        done_message;           ! Success message.

ON_ERROR
    [TPU$_CLIPBOARDLOCKED]:
        eve$message (EVE$_CLIPBDWRITLOCK);
        eve$learn_abort;
        RETURN (FALSE);
    [OTHERWISE]:
        eve$learn_abort;
ENDON_ERROR;
```

```
remove_range := eve$selection (TRUE);
IF remove_range <> 0
THEN
    WRITE_CLIPBOARD ("", remove_range);  ! This statement writes a copy
                                         ! of the selected range to the
                                         ! clipboard.
    IF delete_range
    THEN
        done_message := EVE$_REMCOMPL;
        ERASE (remove_range);
    ELSE
        done_message := EVE$_COPYCOMPL;
    ENDIF;
    remove_range := 0;
    eve$message (done_message);
    RETURN (TRUE);
ENDIF;

eve$learn_abort;
RETURN (FALSE);

ENDPROCEDURE;
```

This procedure shows one possible way that a layered application can use the WRITE_CLIPBOARD built-in. This procedure is a copy of the EVE procedure *eve$$cut_copy*. You can find this procedure in SYS$EXAMPLES:EVE$DECWINDOWS.TPU. For more information on using the files in SYS$EXAMPLES as examples, see Section 7.7.

The procedure checks whether a selection is active and, if so, writes the contents of the selected range to the clipboard. If the user has directed EVE to cut the selected text, the procedure erases the selected range.

WRITE_GLOBAL_SELECT

Sends requested information about a global selection from the VAXTPU layered application to the application that issued the information request.

FORMAT

WRITE_GLOBAL_SELECT ({ *array* | *buffer* | *range* | *string* | *integer* | *NONE* })

PARAMETERS

array

An array passing information about a global selection whose contents describe information that is not of a data type supported by VAXTPU. For example, the array could pass information about a pixmap, an icon, or a span.

VAXTPU does not use or alter the information in the array; the application layered on VAXTPU is responsible for determining how the information is used, if at all. Since the array is used to pass information to and from other DECwindows applications, all applications that will send or receive information whose data type is not supported by VAXTPU must agree on how the information is to be sent and used.

The application sending the information is responsible for creating the array and giving it the proper structure. The array's structure is as follows:

- The element *array {0}* contains a string naming the data type of the information being passed. For example, if the information being passed is a span, the element contains the string "*SPAN*".
- The element *array {1}* contains either the integer 8, indicating that the information is passed as a series of bytes, or the integer 32, indicating that the information is passed as a series of longwords.
- If *array {1}* contains the value 8, the element *array {2}* contains a string and there are no array elements after *array {2}*. The string does not name anything, but rather is a series of bytes. As mentioned, the meaning and use of the information is agreed upon by convention among the DECwindows applications.
- If *array {1}* contains the value 32, the element *array {2}* contains an integer. In this case, the array can have any number of elements after *array {2}*. These elements must be numbered sequentially, starting at *array {3}*. All the elements contain integers. Each integer represents a longword of data. To determine how many longwords are being passed, an application can determine the length of the array and subtract 2 to allow for elements *array {0}* and *array {1}*.

buffer
The buffer containing the information to be sent to the requesting application as the response to the global selection information request. If you specify a buffer, VAXTPU converts the buffer to a string, converts line breaks to line feeds, and inserts padding blanks before text to fill any unoccupied space before the left margin.

range
The range containing the information to be sent to the requesting application as the response to the global selection information request. If you specify a range, VAXTPU converts the buffer to a string, converts line breaks to line feeds, and inserts padding blanks before and after text to fill any unoccupied space before the left margin.

string
The string containing the information to be sent to the requesting application as the response to the global selection information request. VAXTPU sends the information in string format.

integer
An integer whose value is to be sent to the requesting application as the response to the global selection information request. VAXTPU sends the information in integer format.

NONE
A keyword indicating that no information about the global selection is available.

DESCRIPTION

WRITE_GLOBAL_SELECT is valid only inside a routine that responds to requests for information about a global selection.

The parameter specifies the data to supply to the requesting application. If you specify NONE, VAXTPU informs the requesting application that no information is available. Note, however, that for any case in which a routine omits a WRITE_GLOBAL_SELECT statement, by default VAXTPU informs the requesting application that no information is available.

Call WRITE_GLOBAL_SELECT no more than once during the execution of a global selection read routine. It signals TPU$_INVBUILTIN if you attempt to call this routine more than once.

SIGNALED ERRORS

TPU$_BUILTININV	WARNING	WRITE_GLOBAL_SELECT has been used more than once in the same routine.
TPU$_TRUNCATE	WARNING	VAXTPU has truncated characters from the data written because you specified a buffer or range containing more than 65,535 characters.

TPU$_INVPARAM	ERROR	One of the parameters was specified with data of the wrong type.
TPU$_NORETURNVALUE	ERROR	The built-in cannot return a value.
TPU$_REQSDECW	ERROR	You can use the built-in only if the DECwindows screen updater is in use.
TPU$_TOOFEW	ERROR	Too few arguments passed to the built-in.
TPU$_TOOMANY	ERROR	Too many arguments passed to the built-in.

EXAMPLE

The following statement sends the contents of the range *this_range* to the requesting application.

```
WRITE_GLOBAL_SELECT (this_range);
```

For an example of a procedure using the WRITE_GLOBAL_SELECT built-in, see Example 7–11.

6 Writing Code Compatible with DECwindows EVE

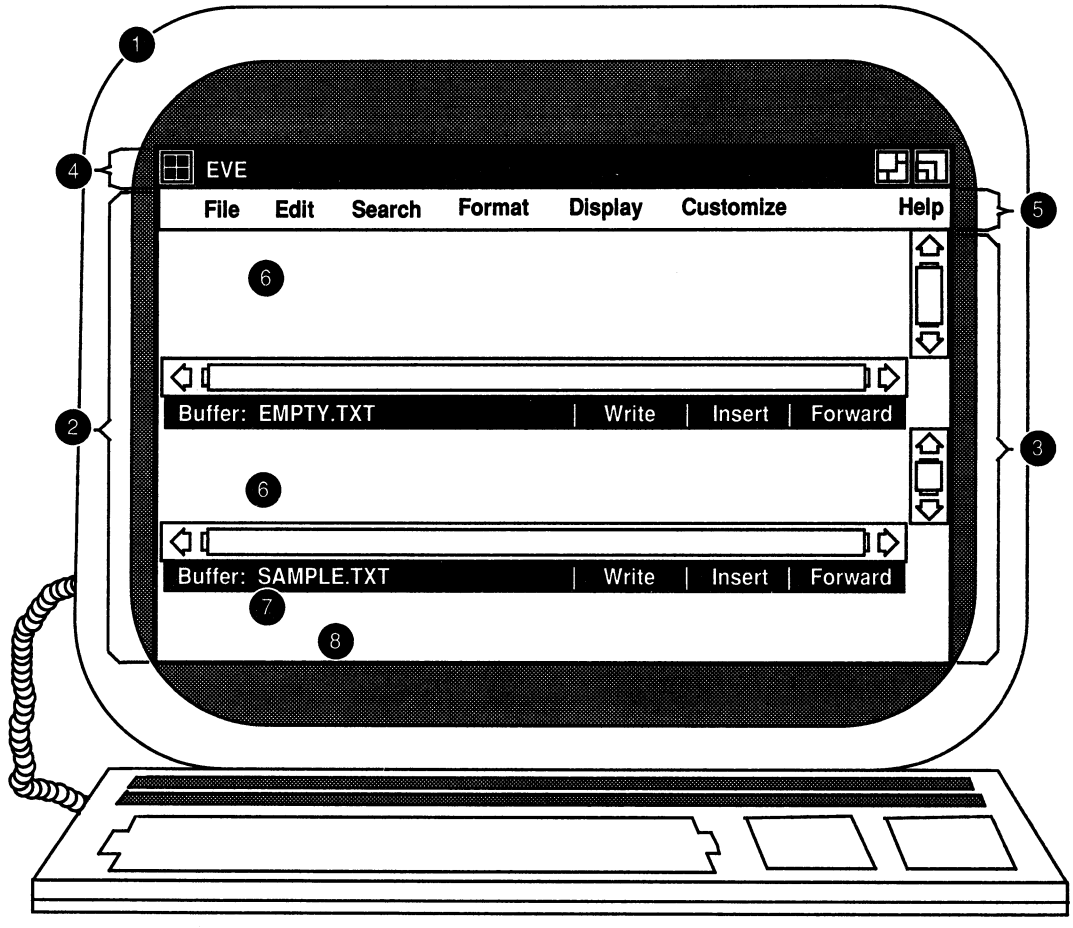
This chapter provides information useful for programmers who extend EVE or layer applications on EVE.

For more information about using the DECwindows EVE editor, see the *VMS Version 5.1 Release Notes*.

6.1 Screen Objects in Applications Layered on DECwindows VAXTPU

Figure 6–1 and its accompanying text show the nomenclature for the screen objects used in EVE and, optionally, in other applications layered on VAXTPU.

Figure 6–1 Nomenclature of DECwindows VAXTPU Screen Objects



ZK–0239A–GE

Key to Figure 6-1

1 Display—In VAXTPU, the term **display** refers to the physical display device on which screen objects are visible.

2 Main window widget—This widget is created by VAXTPU, not by the layered application. Although the main window widget is not visible as a separate entity, it is the ancestor of all of EVE's visible widgets. The VAXTPU SCREEN keyword, when used as a parameter to a widget-related built-in, refers to the main window widget.

VAXTPU's main window widget is associated with a DECwindows window. Both DECwindows and VAXTPU have objects called "windows." VAXTPU windows have much the same function as DECwindows windows, but VAXTPU windows operate within a more limited scope.

A DECwindows window is a viewport enabling a DECwindows application to make visible some text and graphics. For example, a DECwindows window can be used as a viewport onto a widget. A DECwindows window is mapped to an area on a physical display device. For more information about DECwindows windows, see the *VMS DECwindows Guide to Application Programming*.

A VAXTPU window is a viewport onto a VAXTPU buffer. VAXTPU windows always have the same width as the VAXTPU screen. For more information about the VAXTPU screen, see item 3 in this key. You can map a VAXTPU window only within an area of the physical display device occupied by a VAXTPU screen. For more information about mapping VAXTPU windows, see the *VAX Text Processing Utility Manual*.

3 VAXTPU screen—This widget is created by VAXTPU, not by the layered application. When you use the SCREEN keyword as a parameter to a built-in unrelated to widgets, the keyword refers to the VAXTPU screen. In the character-cell version of VAXTPU, the phrase "VAXTPU screen" means all the area visible on the physical terminal screen.

4 Title bar—The title bar for EVE (or any other application layered on VAXTPU) is created by DECwindows, not by VAXTPU or the layered application.

5 Menu bar—The EVE menu bar widget is created by EVE, not by VAXTPU. You can optionally create a menu bar widget in any application layered on VAXTPU. If you do so, make the menu bar widget a child of the VAXTPU main window widget.

6 EVE user window—This window is created by EVE and is mapped to a buffer. It is a VAXTPU window, not a widget. Other applications layered on VAXTPU should create one or more user windows in which to display the results of your actions.

7 EVE command window—This window is created by EVE. It is a VAXTPU window, not a widget. Other applications layered on VAXTPU can optionally create a command window.

8 EVE message window—This window is created by EVE. It is a VAXTPU window, not a widget. Other applications layered on VAXTPU can optionally create a message window.

6.2 Select Ranges in DECwindows EVE

This section is intended for programmers extending EVE or layering an application on EVE.

EVE can use only one type of selection at a time. There are four possible types of selection: dynamic selections, static selections, found range selections, and DECwindows primary global selections. The ways in which these selections differ are explained in the following sections.

EVE has a routine called EVE$SELECTION that returns the current selection, regardless of whether the selection is dynamic, static, or formed from a found range. It is possible to use the VAXTPU built-in SELECT_RANGE to obtain the current selection if the selection is a dynamic selection. However, DIGITAL recommends that you use EVE$SELECTION to obtain the current selection, because this routine returns the current selection regardless of how it was created.

6.2.1 Dynamic Selections

When you press the SELECT key or invoke the EVE command SELECT, EVE creates a dynamic selection. A dynamic selection expands and contracts as you move the text cursor. Moving the text cursor away from the text already selected does not cancel the selection. If you use the mouse to start a selection while a dynamic selection is active, the dynamic selection is canceled.

If EVE's current selection is a dynamic selection, the routine EVE$SELECTION returns the selected range and terminates the selection. If, for some reason, you want to use a statement that returns the current dynamic selection but does not terminate it, you can use a statement whose format is similar to the following:

```
r1 := EVE$SELECTION (TRUE, TRUE, TRUE, TRUE, FALSE)
```

The last parameter directs EVE$SELECTION not to terminate the selection.

6.2.2 Static Selections

EVE creates a static selection if you do any of the following:

- Click the MB1 mouse button two or more times to select a word, line, paragraph, or buffer
- Use the EVE command SELECT ALL
- Press the MB1 mouse button, drag the mouse across text, and then release the mouse button
- Use the MB1 mouse button with the SHIFT key to extend a selection

EVE implements a static selection by creating a range upon which you can perform EVE commands such as STORE TEXT or REMOVE. However, EVE does not start this range using the VAXTPU built-in SELECT. Thus, if you use the SELECT_RANGE built-in while a static selection is active, VAXTPU returns the message "No select active."

If you move the text cursor off the text in the static selection, the selection is canceled.

6.2.3 Found Range Selections

When EVE positions to the beginning of a range as the result of the FIND command, WILDCARD FIND command, or pressing the FIND key, EVE creates a found range containing the text EVE found as a match for your search string. If no dynamic selection is active, EVE treats the found range as the current selection.

EVE implements a found range selection by creating a range upon which you can perform EVE commands such as STORE TEXT or REMOVE. However, EVE does not start this range using the VAXTPU built-in SELECT. Thus, if you use the SELECT_RANGE built-in while a found range selection is active, VAXTPU returns the message "No select active."

If you move the text cursor off the text in the found range selection, the selection is canceled.

6.2.4 Relation of EVE Selection to DECwindows Global Selection

If EVE has a dynamic selection or a static selection active, that selection is automatically designated as the primary global selection. A found range selection is not designated as the primary global selection.

You can use the routine EVE$SELECTION to obtain the text of the primary global selection when an application other than VAXTPU owns the selection. To do so, the call to EVE$SELECTION must be in code bound to a mouse button other than MB1. The value returned is a string containing the text of the primary global selection.

7 Programming in DECwindows VAXTPU

This chapter provides information about various aspects of programming with DECwindows VAXTPU.

7.1 Widgets Supported by DECwindows VAXTPU

DECwindows VAXTPU enables you to create widgets from within VAXTPU programs using the CREATE_WIDGET built-in. For information about how to use widgets to create a DECwindows text processing interface, see the *XUI Style Guide* and the *VMS DECwindows Guide to Application Programming*. For information about the characteristics of specific widgets, see the *VMS DECwindows Toolkit Routines Reference Manual*.

Using the CREATE_WIDGET built-in, you can create the following widgets in VAXTPU:

- Caution_box
- Dialog_box
- File_selection
- Label
- List_box
- Main_window
- Menu_bar
- Popup_attached_db
- Popup_dialog_box
- Popup_menu
- Pulldown_entry
- Pulldown_menu
- Push_button
- Scroll_bar (vertical and horizontal)
- Separator
- Simple_text
- Toggle_button

7.2 Global Selection Support in DECwindows VAXTPU

Global selection in VMS DECwindows is a means of preserving information selected by the user so the user's selection, or data about the user's selection, can be passed between DECwindows applications. Each DECwindows application can own one or more global selections.

7.2.1 Difference Between Global Selection and Clipboard

A global selection differs from the clipboard in that the global selection changes dynamically as the user changes the select range, while the contents of the clipboard remain unchanged until the user uses a command (such as the EVE STORE TEXT command) that sends new information to the clipboard.

7.2.2 Handling of Multiple Global Selections

Each global selection is owned by at most one DECwindows application; a global selection can also be unowned. A DECwindows application can own more than one global selection at the same time. For example, an application layered on VAXTPU can own both the primary and secondary global selection properties. The DECwindows server determines which application owns which global selection.

Information about a global selection property may be stored in different formats, but the format of a particular piece of information must be the same for all DECwindows applications. VAXTPU directly accepts information that is stored in integer or string format. VAXTPU handles information in other formats by describing the information in an array. For more information about this use of an array, see the descriptions of the built-ins GET_GLOBAL_SELECT and WRITE_GLOBAL_SELECT in Chapter 5.

Global selections are identified in VAXTPU either as strings or keywords. While DECwindows provides for many global selections, applications conforming to the *XUI Style Guide* are concerned with only two selections, the **primary** and **secondary** selections. VAXTPU provides a pair of keywords (PRIMARY and SECONDARY) to refer to these selections. VAXTPU also provides built-in procedures that allow layered applications to manipulate global selection information.

You can refer to other global selections by specifying a string instead of the keywords PRIMARY and SECONDARY. For example, if your application has a global selection whose name is *auxiliary*, specify the selection using the string "auxiliary." Note that selection names are case sensitive; the string "auxiliary" does not refer to the same global selection as the string "AUXILIARY."

7.2.3 Relation of Global Selection to Input Focus

An application that conforms to the *XUI Style Guide* requests ownership of the primary global selection in its input focus grab procedure. Regardless of whether the application conforms, when VAXTPU obtains ownership of the input focus, it automatically grabs ownership of the primary global selection if it is not already the owner. An application cannot prevent VAXTPU from attempting to assert ownership of the primary global selection when VAXTPU receives the input focus. If you are attempting to write an application that conforms to the *XUI Style Guide* and you find that VAXTPU has had to grab ownership and execute the global selection routine automatically, your application may have a design problem.

Once VAXTPU owns the primary global selection, it automatically executes the application's global selection grab routine if one is present.

7.2.4 Response to Requests for Information About the Global Selection

VAXTPU provides a three-level hierarchy for responding to requests from another application for information about the current selection. Applications layered on VAXTPU may specify a routine that responds to requests for information about global selections either for the entire application or for one or more buffers in the application. When VAXTPU receives a request for information, it checks whether there is a routine for the current buffer that responds to information about global selections. If no buffer-specific routine is available, VAXTPU checks for an application-wide routine. If no application-wide routine is available, VAXTPU attempts to respond to the request itself, but it can only respond to requests for information about the primary selection and only provides information about the file name, font, line number, and text. VAXTPU responds to all other requests with a message that no information is available. Note that VAXTPU does not send requests for information about the global selection to other DECwindows applications.

VAXTPU's responses to requests for information about the primary selection are as follows:

"FILE_NAME"	VAXTPU responds with the string returned by the built-in procedure GET_INFO (CURRENT_BUFFER, "file_name").
"FONT"	VAXTPU responds with the string returned by the built-in procedure GET_INFO (SYSTEM, "default_font").
"LINE_NUMBER"	VAXTPU responds with value of type SPAN containing the record number where the select range starts and the record number where the select range ends.
"TEXT" or "STRING"	VAXTPU responds with the text of the select range as a string, with each line break represented by a line feed.

DIGITAL recommends that you not use a non-DECwindows section file with the DECwindows version of VAXTPU. However, if you do not follow this recommendation, VAXTPU's automatic grabbing of the primary global selection allows your layered application to interact with other DECwindows applications. If an application requests information about the primary global selection while VAXTPU owns the selection, VAXTPU attempts to respond to the request if the application cannot do so. If

VAXTPU responds to the request by sending the text of a buffer or range, VAXTPU converts the buffer or range to a string, converts line breaks to line feeds, and inserts padding blanks before text to fill any unoccupied space between the margins. If neither the application nor VAXTPU can respond to the request, VAXTPU informs DECwindows that the requested information is not available.

VAXTPU does not automatically grab the secondary selection. Layered applications are responsible for handling this selection.

7.3 Input Focus Support in DECwindows VAXTPU

In VMS DECwindows, at most one of the applications on the screen can have the **input focus**; that is, an application can have the ability to accept user input from the keyboard. For more information about the input focus, see the *XUI Style Guide*.

DECwindows VAXTPU automatically grabs the input focus whenever the user causes an unmodified M1DOWN event (that is, an event not modified by SHIFT, CTRL, or other modifying key) while the pointer cursor is in either of the following locations:

- VAXTPU's main window widget
- VAXTPU's title bar

When DECwindows VAXTPU grabs input focus or when an application layered on VAXTPU requests input focus, DECwindows assigns the focus to VAXTPU only if and when it is possible to do so. Therefore, your application should use the GET_INFO (SCREEN, "input_focus") built-in to test whether it actually has the input focus before performing any operation that requires the input focus.

DIGITAL recommends that you not use a non-DECwindows section file with the DECwindows version of VAXTPU. However, if you do not follow this recommendation, VAXTPU's automatic grabbing of the input focus allows your layered application to interact with other applications in a DECwindows environment.

7.4 Using Callbacks in DECwindows VAXTPU

When the user performs an action affecting a widget, the widget sends a **callback**, or notification, to the application of which the widget is a part. The callback is sent to a **callback routine**, which is the portion of the application that defines what the application does in response to the callback. For more information about the use of callbacks and callback routines in DECwindows programs, see the *VMS DECwindows Guide to Application Programming*.

There are many possible ways to design an application layered on DECwindows VAXTPU. Therefore, there is no single "right" way to handle callbacks when creating a layered application. However, the following information describing the relationship between the VAXTPU engine and a layered application is helpful for the programmer who is layering an application with a DECwindows interface on VAXTPU.

Creating an application layered on VAXTPU can involve the following different levels of mechanisms for handling callbacks:

- Callable interface-level callback routines—Routines internal to VAXTPU that dispatch a widget's callbacks to the layered application
- Application-level callback programs or learn sequences—Layered application code that is executed after VAXTPU has received a callback from a widget

At the callable interface level, VAXTPU has several callback routines, one of which may be called by a widget when an event occurs. An event is a user action or other occurrence that affects a widget. Note that although DECwindows allows you to specify a different callback routine for each reason that a widget can call back, DECwindows VAXTPU does not support this capability. For more information about engine-level callback routines, see Section 7.4.1.

At the application level, an application contains one or more callback programs or learn sequences specified with CREATE_WIDGET or SET (WIDGET_CALLBACK). VAXTPU executes the program or learn sequence when it receives a callback from a widget. For more information about application-level callback routines, see Section 7.4.2.

7.4.1 Callable Interface-Level Callback Routines

If you are layering an application on VAXTPU or on EVE, you specify callable interface-level callback routines only if you are specifying a widget's callback resources in a User Interface Language (UIL) file.

Callbacks can pass values known as **closures**, which are strings or integers whose function depends on the application you are writing. For more information about what closures are and how to use them, see Section 7.5.

You use the VAXTPU callable interface routine TPU$WIDGET_INTEGER_CALLBACK as the callback routine for all callbacks that have an integer closure and the VAXTPU routine TPU$WIDGET_STRING_CALLBACK for all callbacks that have a string closure.

Although the SET (WIDGET) built-in allows you to specify values for various resources of a widget, there are restrictions on specifying values for callback resources of widgets not organized in an XUI Resource Manager hierarchy. When a widget is part of an XUI Resource Manager hierarchy, do not include callback resource names or values in the array you pass to SET (WIDGET). Instead, specify the callback routine in the UIL file. When a widget is not part of an XUI Resource Manager hierarchy, use the names of the callback resources in the array you pass to SET (WIDGET), and specify 0 as the value of each such callback resource. VAXTPU automatically substitutes its common callback entry point for the 0 value. Note that a widget that is not part of an XUI Resource Manager hierarchy calls back only for those reasons specified in the widget's argument list. If a reason is omitted from the list, the corresponding event does not cause a callback.

7.4.2 Application-Level Callback Programs or Learn Sequences

When you specify an application-level callback program or learn sequence with CREATE_WIDGET or SET (WIDGET_CALLBACK), all widgets in the same XUI Resource Manager hierarchy have the same callback program or learn sequence. Therefore, the callback program or learn sequence must have a mechanism for handling all possible callback reasons.

7.5 Using Closures in DECwindows VAXTPU

DECwindows allows you to specify a closure value for a widget. DECwindows does not define what a closure value is; a closure is simply a value that DECwindows understands how to recognize and manipulate so that a DECwindows application programmer can use the value if needed in the application. For general information about using closures in DECwindows, see the *VMS DECwindows Guide to Application Programming*.

When a widget calls back to the DECwindows application, the callback parameters include the closure value assigned to the widget. DECwindows allows the application to define the significance and possible values of the closure.

VAXTPU supports closure values of type STRING and INTEGER. Closure values are optional for widgets used by applications layered on VAXTPU. If you do not specify a closure value, the built-in GET_INFO (WIDGET, "callback_parameters", array) returns UNSPECIFIED in the "*closure*" array element. If you create a widget without using a UIL file, the built-in GET_INFO (WIDGET, "callback_parameters", array) returns the closure you specified as a parameter to CREATE_WIDGET. If you create a widget using a UIL file, the built-in GET_INFO (WIDGET, "callback_parameters", array) returns the closure value (if any) defined in the XUI Resource Manager. If none is defined, the built-in returns UNSPECIFIED.

VAXTPU leaves it to the layered application to use the closure resource in any way the application programmer wishes. VAXTPU passes through to the application any closure value received as part of a callback.

The DECwindows version of EVE provides an example of how an application can use closure values. DECwindows EVE assigns a unique closure value to every widget instance that can be created during an EVE editing session. Each closure value corresponds to something that EVE must do in response to the activation of that particular widget. When an event causes VAXTPU to execute EVE's main callback program, the built-in GET_INFO (WIDGET, "callback_parameters", array) returns the widget activated, the reason code (the reason the widget is calling back), and the closure associated with the particular widget instance. EVE's main callback program contains an array that is indexed with values identical to the widget closure values. Each array element contains a pointer to the EVE code to be executed in response to the corresponding widget's callback. EVE's callback program uses the closure value to locate the appropriate array index so the correct EVE routine can be executed in response to the callback.

If your layered application does not use EVE's callback program, then its callback program or learn sequence must have a mechanism for determining which widget is calling back and which application code should be executed as a result.

7.6 Specifying Values for Widget Resources in DECwindows VAXTPU

This section discusses techniques for specifying values for widget resources.

7.6.1 VAXTPU Data Types for Specifying Resource Values

VAXTPU supports the following data types with which to specify values for widget resources:

- String
- Array of strings
- Integer

VAXTPU converts the value you specify into the data type appropriate for the widget resource you are setting. The following table shows the relationship between VAXTPU data types for widget resources and DECwindows data types for widget resources:

Table 7–1 Correspondence Between VAXTPU Data Types and DECwindows Argument Data Types

DECwindows Argument Data Type	VAXTPU Data Type
Array of strings	Array of strings
Boolean	Integer
Callback	Integer (0)
Compound string	String
Compound string table	Array of strings
Dimension	Integer
Integer	Integer
Position	Integer
Short	Integer
String	String
Unsigned character	Integer

VAXTPU does not support setting values for resources (such as pixmap, colormap, font, icon, and so on) whose data types are not listed in this table.

When you pass an array specifying values for a widget's resources using CREATE_WIDGET or SET (WIDGET), VAXTPU verifies that each array index is a string corresponding to a valid resource name for the specified

widget. VAXTPU also verifies that the data type of the value you specify is valid for the specified resource.

7.6.2 Specifying a List as a Resource Value

List box and file selection widgets manipulate lists. For example, the file selection widget manipulates a list of files. The widget resource that stores such a list is specified to VAXTPU using an array.

To handle an array that passes a list to a widget, DECwindows must know how many elements the array contains. For example, if you, the application programmer, set the value of the "item" resource of a list box widget to point to a given array, DECwindows does not handle the array successfully unless the list box widget's "itemsCount" resource contains the number of elements in the array.

However, you do not necessarily know how many elements the array has at a given moment. To help you pass arrays, VAXTPU has a convention for referring to widget resources. If you follow the convention, VAXTPU will handle the resource that stores the number of array elements. The following paragraphs discuss the naming convention in more detail.

When you use the VAXTPU built-in procedure SET (WIDGET) to pass a list to a widget, specify both the list name and the list count resource in the same array index, separated by a line feed (ASCII (10)). The array element should be the array that is to be passed. For example, to specify the "items" resource to the list box widget, use code similar to the following:

```
line_feed := ASCII (10);
resource_array {"items" + line_feed + "itemsCount"}:=list_array;
```

The line-feed character, ASCII (10), is a delimiter separating two resource names.

VAXTPU automatically generates two resource entries. The first is the array of strings specifying the data to the list box for the "items" resource. The second is the count of elements in the array for the "itemsCount" resource.

To get resource values from a widget, use the following statement:

```
GET_INFO (widget, "WIDGET_INFO", array)
```

The indices of the array parameter are strings or string constants naming the resources whose value you want. (The initial values in the array are unimportant.) The GET_INFO statement directs VAXTPU to fetch the specified resource values of the specified widget and put the values in the array.

For list box widget or file selection widgets, one element of the array receives another array containing the list manipulated by the widget. When you create the index of the element that receives the widget's list, you must observe the naming convention so that VAXTPU can handle both the list itself and the resource value specifying the length of the list. Give the index the following format:

```
items<line-feed>items_count
```

For example, if you used GET_INFO (widget, "WIDGET_INFO", array) to get resource values from a list box widget, you could specify the index for the element storing the widget's list as follows:

```
"items" + ASCII(10) + "itemsCount"
```

Note that the element for the widget's list does not actually contain an array until after execution of the GET_INFO statement. When VAXTPU encounters the GET_INFO statement, it parses the indices of the specified array. When VAXTPU parses the index of the element for the widget's list, it fetches both the list itself and the length of the list. Using the resource specifying the length, VAXTPU creates an array of the correct size to hold the widget's list.

7.7 Sample Uses of DECwindows VAXTPU Built-Ins

You can use the DECwindows VAXTPU built-in procedures in many ways. However, you may find it useful to look at sample procedures showing how other programmers have used some of the DECwindows VAXTPU built-ins. Therefore, this section presents a number of procedures using DECwindows built-ins. Most of the procedures are drawn from the code implementing the Extensible VAX Editor (EVE). Some have been modified to make them easier to understand.

You can see all the code used to implement EVE by looking at the files in the directory pointed to by the logical name SYS$EXAMPLES. To see a directory of the files available, type the following command from the DCL command line:

```
$ DIR SYS$EXAMPLES:EVE$*.TPU
```

These files contain procedures using almost all of the new and modified built-ins.

7.7.1 Example of Displaying a Dialog Box

Example 7–1 illustrates one of the ways a layered application can use the CONVERT built-in. This procedure is a modified version of the EVE procedure *eve$$mb2_dispatch*. You can find the original version in SYS$EXAMPLES:EVE$MOUSE.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section 7.7.

The procedure displays EVE's selection popup menu on the screen if the procedure is called while a select range or found range is active.

This example uses the following global variables and procedures:

- EVE$CALLBACK_DISPATCH - The procedure that EVE uses to dispatch all widget callbacks.
- EVE$X_FOUND_RANGE - A global variable that holds the range for the last text found. If there is not currently a found range, it is set to zero.

- EVE$X_SELECT_POPUP - A global variable that holds the popup menu widget used when a selection is present.
- EVE$X_SELECT_POPUP_HEIGHT - A global variable that holds the height of the selection popup menu.
- EVE$X_SELECT_POPUP_WIDTH - A global variable that holds the width of the selection popup menu.
- EVE$X_SELECT_POSITION - A global variable that holds the start marker for the select range. If there is not currently a selection, it is set to zero.

Example 7–1 EVE Procedure Displaying a Selection Dialog Box

```
PROCEDURE eve$$mb2_dispatch

local   status,
        the_window,
        temp_array,
        the_widget,
        x_1,
        X2,
        widget_hierarchy,
        y_1,
        y_2;

❶ IF (LOCATE_MOUSE (the_window, x_1, y_1) <> 0)
  THEN
❷     CONVERT (the_window, CHARACTERS, x_1, y_1,
                DECW_ROOT_WINDOW, COORDINATES,
                X2, y_2);

      IF (eve$x_select_position <> 0) OR          ! A selection exists
         (eve$x_found_range <> 0)                 ! A found range exists

      THEN
          IF GET_INFO (eve$x_select_popup, "type") <> WIDGET

          THEN
❸             widget_hierarchy := SET (DRM_HIERARCHY, "EVE$WIDGETS");

              eve$x_select_popup := CREATE_WIDGET ("SELECT_POPUP",
                                                   widget_hierarchy,
                                                   SCREEN,
                                                   "eve$callback_dispatch");

          ENDIF;

              ! Get width and height of this pop-up menu if needed

❹         temp_array := CREATE_ARRAY;
              temp_array {eve$dwt$c_width} := 0;
              temp_array {eve$dwt$c_height} := 0;
              status := GET_INFO (eve$x_select_popup, "WIDGET_INFO", temp_array);
              eve$x_select_popup_width := temp_array {eve$dwt$c_width};
              eve$x_select_popup_height := temp_array {eve$dwt$c_height};
```

(continued on next page)

Example 7–1 (Cont.) EVE Procedure Displaying a Selection Dialog Box

```
        ! Calculate position for upper left corner of
        ! dialog box and set the appropriate resources of the widget

        temp_array := CREATE_ARRAY;
        temp_array {eve$dwt$c_nx} := X2 - (eve$x_select_popup_width/2);
        IF temp_array {eve$dwt$c_nx} < 1
        THEN
            temp_array {eve$dwt$c_nx} := 1;
        ENDIF;
        temp_array {eve$dwt$c_ny} := y_2 - (eve$x_select_popup_height/2);
        IF temp_array {eve$dwt$c_ny} < 1
        THEN
            temp_array {eve$dwt$c_ny} := 1;
        ENDIF;

❺       SET (WIDGET, eve$x_select_popup, temp_array);
❻       MANAGE_WIDGET (eve$x_select_popup);
    ENDIF;
ENDIF;
RETURN (TRUE);

ENDPROCEDURE;
```

❶ The return value from the LOCATE_MOUSE built-in procedure indicates whether the pointer cursor is in the window. LOCATE_MOUSE also returns the row, column, and window where the pointer cursor is located. The coordinates returned refer to a system whose origin is in the upper left corner of the VAXTPU window.

❷ This clause converts the pointer cursor location from a system whose origin is at the upper left corner of the VAXTPU window to a system whose origin is at the upper left corner of the DECwindows root window. For more information about the difference between VAXTPU windows and DECwindows windows, see Chapter 6.

❸ SET (DRM_HIERACHY, file_spec) allows you to tell VAXTPU which XUI Resource Manager hierarchy to use. An XUI Resource Manager hierarchy is a set of widgets implementing a user interface. For example, EVE's menu bar and menu widgets compose an XUI Resource Manager hierarchy.

EVE uses the XUI Resource Manager hierarchy stored in the file EVE$WIDGETS.UID. If you are extending EVE, you need not set the hierarchy again.

VAXTPU allows you to use multiple XUI Resource Manager hierarchies. If you want to use a second hierarchy (defined in a file other than EVE$WIDGETS.UID), use the SET (DRM_HIERARCHY) statement before using the CREATE_WIDGET statement.

❹ GET_INFO (widget, "widget_info", array) allows you to fetch information about a widget. The index of each element of the array must be a string naming the resource whose value you want to fetch. For more information about what resources a given widget supports, see *VMS DECwindows Toolkit Routines Reference Manual*.

(continued on next page)

Example 7–1 (Cont.) EVE Procedure Displaying a Selection Dialog Box

❺ SET (WIDGET, widget, array) allows you to set a widget's resource values. The index of each element of the array must be a string naming the resource whose values you want to set. For more information about what resources a given widget supports, see *VMS DECwindows Toolkit Routines Reference Manual.*

❻ MANAGE_WIDGET realizes the widget and makes it visible on the screen.

7.7.2 Example of Creating a "Mouse Pad"

Example 7–2 shows how to use the variant of CREATE_WIDGET that calls the XUI Toolkit low-level creation routine. The module in Example 7–2 creates a screen representation of a keypad. Instead of pressing a keypad key, a user can click on the widget representing the key.

Example 7–2 Procedure Creating a "Mouse Pad"

```
! SAMPLE.TPU

!++
!                                Table of Contents
!
!                                   SAMPLE.TPU
!
!        Procedure name          Description
!        --------------          -----------
!
!        sample_sample_module_ident   Ident.
!        sample_sample_module_init    Initializes the module.
!        eve_mouse_pad                Implements the user command DISPLAY MOUSE PAD.
!        sample_key_def               Creates a mouse pad "key" push button.
!        sample_key_dispatch          Handles push button widget callbacks.
!        sample_row_to_pix            Converts a row number to pixels.
!        sample_col_to_pix            Converts a column number to pixels.
!        sample_key_height            Converts Y dimension from rows to pixels.
!        sample_key_width             Converts X dimension from columns to pixels.
!--
```

(continued on next page)

Example 7–2 (Cont.) Procedure Creating a "Mouse Pad"

```
! This module layers a "mouse pad" on top of VAXTPU.  The mouse pad
! is implemented by creating a dialog box widget that is the parent of a group
! of push button widgets depicting keypad "keys."  The resulting
! "mouse pad" is a screen representation of a keypad.  The user can
! click on a push button to execute the same function that would be
! executed by pressing the corresponding keypad key.  The module uses
! the key map list mapped to the current buffer to determine what
! code to execute when the user clicks on a given push button.  To
! use a different key map, substitute a string naming the desired
! key map for the null string assigned to "sample_k_keymap".
! This module can be used with the EVE section file
! or with a non-EVE section file.
!
! This module uses the variant of CREATE_WIDGET that calls the XUI
! Toolkit low-level creation routine.


PROCEDURE sample_sample_module_ident                ! This procedure returns
RETURN "V01-001";                                   ! the Ident.
ENDPROCEDURE;

PROCEDURE sample_sample_module_init          ! Module initialization.
ENDPROCEDURE;

! VAXTPU Declarations for XUI Toolkit constants

       ! Use these constants as arguments to the DEFINE_WIDGET built-in.
       ! The strings are the symbols that evaluate to the
       ! widget class records for the DECwindows widgets.

CONSTANT
    sample_k_labelwidgetclass := "labelwidgetclassrec",
    sample_k_dialogwidgetclass := "dialogwidgetclassrec",
    sample_k_push-buttonwidgetclass := "pushbuttonwidgetclassrec";

       ! Use these constants, which are XUI Toolkit
       ! resource name strings, as callback reasons, resource values, or
       ! arguments to the CREATE_WIDGET built-in.

CONSTANT
    sample_k_cstyle := "style",
    sample_k_modeless := 2,
    sample_k_nunits := "units",
    sample_k_pixelunits := 1,
    sample_k_ntitle := "title",
    sample_k_nx := "x",
    sample_k_ny := "y",
    sample_k_nheight := "height",
    sample_k_nwidth := "width",
    sample_k_nlabel := "label",
    sample_kt_nactivate_callback := "activateCallback",
    sample_kt_nborderwidth := "borderWidth",
    sample_kt_nconformToText := "conformToText",
    sample_k_cractivate := 10;

! These constants are intended for use only in this sample module
! because their values are specific to the mouse pad application.
```

(continued on next page)

Example 7–2 (Cont.) Procedure Creating a "Mouse Pad"

```
CONSTANT
    sample_k_x_pos := 500,            ! Screen position for mouse pad.
    sample_k_x_pos := 500,
    sample_k_keypad_border := 5,      ! Width of border between keys and edge.
    sample_k_key_height := 30,        ! Key dimensions.
    sample_k_key_width := 60,
    sample_k_button_border_frac := 3, ! Determines spacing between keys.

    sample_k_overall_height := (sample_k_key_height * 5)
                               + ((sample_k_key_height
                                   / sample_k_button_border_frac) * 5)
                               + sample_k_keypad_border,

    sample_k_overall_width := (sample_k_key_width * 4)
                              + ((sample_k_key_width
                                  / sample_k_button_border_frac) * 4)
                              + sample_k_keypad_border,

    sample_k_keymap := '',       ! If this constant has a null string
                                 ! as its value, the program uses the
                                 ! current key map list to determine what
                                 ! code to execute when the user
                                 ! clicks on a given push button.

    sample_k_pad_title := "Sample mouse pad",    ! Title of the mouse pad.
    sample_k_closure := '';                      ! Not currently used.

PROCEDURE eve_mouse_pad          ! Implements the EVE user command MOUSE PAD.
ON_ERROR
    [TPU$_CONTROLC]:
        eve$learn_abort;
        ABORT;
ENDON_ERROR

! Checks whether the dialog box widget class has already been defined.
! If not, defines the dialog box widget class and creates a widget
! instance to be used as the "container" for the mouse pad.

IF GET_INFO (sample_x_dialog_class, 'type') <> INTEGER
THEN
    sample_x_dialog_class
❶       := DEFINE_WIDGET_CLASS (sample_k_dialogwidgetclass,
                                "dwt$dialog_box_popup_create");
ENDIF;

❷ sample_x_keypad := CREATE_WIDGET (sample_x_dialog_class, "Keypad", SCREEN,
                                    "MESSAGE('CALLBACK activated')",
                                    "sample_k_closure ",
                                    sample_k_cstyle, sample_k_modeless,
                                    sample_k_nunits, sample_k_pixelunits,
                                    sample_k_ntitle, sample_k_pad_title,
                                    sample_k_nheight, sample_k_overall_height,
                                    sample_k_nwidth, sample_k_overall_width,
                                    sample_k_nx, sample_k_x_pos,
                                    sample_k_ny, sample_k_y_pos);

! Checks whether the push-button widget class has already been defined
! and, if not, defines the class.
```

(continued on next page)

Example 7–2 (Cont.) Procedure Creating a "Mouse Pad"

```
IF GET_INFO (sample_x_pushbutton_class, 'type') <> INTEGER
THEN
    sample_x_pushbutton_class                                    ! This statement
        := DEFINE_WIDGET_CLASS (sample_k_pushbuttonwidgetclass, ! using the built_in
                                "dwt$push_button_create");       ! DEFINE_WIDGET_CLASS
ENDIF;                                                           ! defines the
                                                                 ! class of the
                                                                 ! push button
                                                                 ! widgets

! Initializes the array that the program passes repeatedly
! to the procedure "sample_key_def".

sample_x_attributes := CREATE_ARRAY;
sample_x_attributes {sample_k_nactivate_callback} := 0;
sample_x_attributes {sample_k_nborderwidth} := 2;
sample_x_pad_program := COMPILE ("sample_key_dispatch");

! Creates and manages all of the "keys" in the mouse pad.  The procedure
! "sample_key_def" returns a variable of type widget, so you can use the
! returned value as an argument to the built-in MANAGE_WIDGET.

❸ MANAGE_WIDGET (sample_key_def ("PF1", 0, 0, 1, 1, sample_x_pad_program),
                 sample_key_def ("PF2", 1, 0, 1, 1, sample_x_pad_program),
                 sample_key_def ("PF3", 2, 0, 1, 1, sample_x_pad_program),
                 sample_key_def ("PF4", 3, 0, 1, 1, sample_x_pad_program),
                 sample_key_def ("KP7", 0, 1, 1, 1, sample_x_pad_program),
                 sample_key_def ("KP8", 1, 1, 1, 1, sample_x_pad_program),
                 sample_key_def ("KP9", 2, 1, 1, 1, sample_x_pad_program),
                 sample_key_def ("-", 3, 1, 1, 1, sample_x_pad_program, "minus"),
                 sample_key_def ("KP4", 0, 2, 1, 1, sample_x_pad_program),
                 sample_key_def ("KP5", 1, 2, 1, 1, sample_x_pad_program),
                 sample_key_def ("KP6", 2, 2, 1, 1, sample_x_pad_program),
                 sample_key_def (",", 3, 2, 1, 1, sample_x_pad_program, "comma"),
                 sample_key_def ("KP1", 0, 3, 1, 1, sample_x_pad_program),
                 sample_key_def ("KP2", 1, 3, 1, 1, sample_x_pad_program),
                 sample_key_def ("KP3", 2, 3, 1, 1, sample_x_pad_program),
                 sample_key_def ("Enter", 3, 3, 2, 1, sample_x_pad_program,
                                 "enter"),
                 sample_key_def ("KP0", 0, 4, 1, 2, sample_x_pad_program),
                 sample_key_def (".", 2, 4, 1, 1, sample_x_pad_program,
                                 "period"));

sample_x_shift_was_last := FALSE;      ! The program starts out assuming that
                                       ! no GOLD key has been pressed.

❹ MANAGE_WIDGET (sample_x_keypad);     ! This statement displays the
                                       ! resulting mouse pad.

RETURN (TRUE);
ENDPROCEDURE ! End of procedure eve_mouse_pad.

PROCEDURE sample_key_def                 ! Creates a mouse pad "key" push-button
                                         ! widget.
    (the_legend,              ! What characters to show on the push-button label.
     the_row, the_col,        ! Location of the key in relation to the parent
                              ! widget's upper left corner.
```

(continued on next page)

Example 7–2 (Cont.) Procedure Creating a "Mouse Pad"

```
     the_width, the_height,  ! Dimensions of the key.
     the_pgm;                ! Program to use as the callback routine; used
                             ! as a parameter to the CREATE_WIDGET built-in.
     the_string);            ! The string representation of the name
                             ! of a key if the key name is not going
                             ! to be the same as the legend (as in
                             ! the case of the comma).  Specify the null
                             ! string if the key name and the legend are
                             ! the same.
IF GET_INFO (the_string, 'type') = UNSPECIFIED
THEN
    the_string := the_legend;     ! Determines whether the optional parameter
                                  ! the_string is provided.
ENDIF;

RETURN CREATE_WIDGET (sample_k_pushbutton_class, "Key", sample_x_keypad,
                      the_pgm,
                      (sample_k_keymap + ' ' + the_string),
                      sample_x_attributes,
                      sample_kt_nconformToText, 0,
                      sample_k_nlabel, the_legend,
                      sample_k_nheight, sample_key_height (the_width),
                      sample_k_nwidth, sample_key_width (the_height),
                      sample_k_nx, sample_col_to_pix (the_row),
                      sample_k_nx, sample_row_to_pix (the_col));

ENDPROCEDURE ! End of the procedure "sample_key_def".

PROCEDURE sample_key_dispatch  ! Handles push-button widget callbacks.

LOCAL   status,                ! Variable to contain the return value from
                               ! GET_INFO (WIDGET, "callback_parameters",).
        blank_index,           ! Position of the blank space in the tag string.
        temp_array,            ! Holds callback parameters.
        a_shift_key,           ! The SHIFT key in the current key map list.
        the_key,               ! A string naming a key.
        gold_key;              ! Name of the GOLD key.

ON_ERROR
    [TPU$_CONTROLC]:
        eve$learn_abort;
        ABORT;
ENDON_ERROR

❺ status := GET_INFO (widget, "callback_parameters", temp_array);
  $widget := temp_array {'widget'};
  $widget_tag := temp_array {'closure'};
  $widget_reason := temp_array {'reason_code'};

❻ the_key := EXECUTE ("RETURN(KEY_NAME (" + $widget_tag + "))");
  gold_key := GET_INFO (eve$current_key_map_list, "shift_key");
  IF the_key = gold_key
  THEN
      sample_shift_was_last := TRUE;      ! User pressed Gold Key
```

(continued on next page)

Example 7–2 (Cont.) Procedure Creating a "Mouse Pad"

```
ELSE
    IF sample_shift_was_last
    THEN
        the_key := KEY_NAME (the_key, SHIFT_KEY);

    ENDIF;
    CASE $widget_reason
        [sample_kt_cractivate]:
            EXECUTE (the_key);
        [OTHERWISE]:
            eve_show_key (the_key)
    ENDCASE;
    sample_shift_was_last := FALSE;
ENDIF;
RETURN;
ENDPROCEDURE ! End of the procedure "sample_key_dispatch".

! These procedures implement position and
! size calculations for the push-button widgets.

PROCEDURE sample_row_to_pix (row)        ! Converts a row number to the
                                         ! pixel-based measuring system.
RETURN sample_k_keypad_border +
    (row * (sample_k_key_height + (sample_k_key_height
                                   / sample_k_button_border_frac)));
ENDPROCEDURE ! End of the procedure "sample_row_to_pix".

PROCEDURE sample_col_to_pix (col)       ! Converts a column number to the
                                        ! pixel-based measuring system.
RETURN sample_k_keypad_border +
    (col * ((sample_kt_key_width + sample_kt_key_width)
                                  / sample_kt_button_border_frac ));

ENDPROCEDURE ! End of the procedure "sample_col_to_pix".

PROCEDURE sample_key_height (given_height) ! Converts the Y dimension
                                           ! from rows to pixels.
IF given_height = 1
THEN
    RETURN sample_k_key_height;
ELSE
    RETURN ((sample_k_key_height * given_height)
            + (sample_k_key_height / sample_k_button_border_frac)
            * (given_height - 1));
ENDIF;
ENDPROCEDURE ! End of the procedure "sample_key_height".

PROCEDURE sample_key_width (given_width)    ! Converts the X dimension
                                            ! from rows to pixels.
IF given_width = 1
THEN
    RETURN sample_k_key_width;
ELSE
    RETURN ((sample_k_key_width * given_width)
            + (sample_k_key_width / sample_k_button_border_frac)
            * (given_width - 1));
ENDIF;
ENDPROCEDURE ! End of the procedure "sample_key_width".
```

(continued on next page)

Example 7–2 (Cont.) Procedure Creating a "Mouse Pad"

❶ When you create widgets directly in VAXTPU (that is, without using the XUI Resource Manager to manipulate widgets defined in a UIL file) you must define each **class** of widget. For example, a widget can belong to the class of push button widgets, dialog box widgets, menu widgets, or another similar class. The DEFINE_WIDGET_CLASS built-in procedure tells VAXTPU the widget class record name and creation entry point for the class of widget. DEFINE_WIDGET_CLASS also returns a widget id for that widget class. Define a widget class for each class of widget only once in a VAXTPU session.

❷ The CREATE_WIDGET built-in allows you to create an **instance** of a widget for which you have a widget id. An instance is one occurrence of a widget of a given class. For example, EVE has many menu widgets, each of which is an instance of a menu widget.

This example creates a dialog box widget to contain the mouse pad.

❸ Each of the keys of the mouse pad is managed. However, they do not become visible until their parent, the dialog box widget in variable SAMPLE_X_KEYPAD, is managed.

❹ Managing a widget whose parent is visible causes that widget and all its managed children to become visible.

❺ GET_INFO (WIDGET, "callback_parameters", array) returns the callback information in the array parameter. For more information about using this built-in, see the built-in's description in Chapter 5.

❻ When each key widget of the mouse pad was created, the closure value for the widget was set to be the string corresponding to the name of the key that the widget represents. This statement uses the EXECUTE built-in to translate the string into a key name.

7.7.3 Example of Implementing an EDT-Style APPEND Command

Example 7–3 shows one of the ways an application can use the GET_CLIPBOARD built-in. This procedure is a modified version of the EVE procedure *eve$edt_append*. You can find the original version in SYS$EXAMPLES:EVE$EDT.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section 7.7.

The procedure *eve$edt_append* appends the currently selected text to the contents of the clipboard if the user has activated the clipboard; otherwise, the procedure appends the current selection to the contents of the Insert Here buffer.

This example uses the following global variables and procedures from EVE:

- EVE$MESSAGE - A procedure that translates the specified message code into text and displays the text in the message buffer.

- EVE$$RESTORE_POSITION - A procedure that repositions the editing point to the location indicated by the specified window and marker. This procedure is for EVE internal use only. Do not call this procedure in a user-written procedure.
- EVE$LEARN_ABORT - A procedure that aborts a learn sequence.
- EVE$SELECTION - A procedure that returns a range containing the current selection. This can be the select range, the found range, or the text of the global selection.
- EVE$$TEST_IF_MODIFIABLE - A procedure that checks whether a buffer can be modified. This procedure is for EVE internal use only. Do not call this procedure in a user-written procedure.
- EVE$X_DECWINDOWS_ACTIVE - A Boolean global variable that is true if VAXTPU is using the DECwindows screen manager. If VAXTPU is not using the DECwindows screen manager, the DECwindows features are not available.
- EVE$$X_STATE_ARRAY - A global variable of type ARRAY describing various EVE flags and data. This variable is private to EVE and should not be used by user routines.
- EVE$$EDT_APPEND_PASTE - Procedure which appends text to the Insert Here buffer. This procedure is for EVE internal use only. Do not call this procedure in a user-written procedure.

Example 7–3 EVE Procedure Implementing a Variant of the EDT APPEND Command

```
PROCEDURE eve$edt_append          ! Implements EVE's version of
                                  ! the EDT APPEND command.
LOCAL   saved_mark,               ! Marks the editing point at the
                                  ! beginning of the procedure.

        remove_range,             ! Stores the currently selected text.

        old_string,               ! Stores the text that was in the clipboard.

        new_string,               ! Stores the old contents of the clipboard
                                  ! plus the currently selected text.

        remove_status;            ! Indicates whether the selected text
                                  ! should be removed.
ON_ERROR
    [TPU$_CLIPBOARDNODATA]:
        eve$message (EVE$_NOINSUSESEL);
        eve$$restore_position (saved_mark);
        eve$learn_abort;
        RETURN (FALSE);
    [TPU$_CLIPBOARDLOCKED]:
        eve$message (EVE$_CLIPBDREADLOCK);
        eve$$restore_position (saved_mark);
        eve$learn_abort;
        RETURN (FALSE);
```

(continued on next page)

Example 7–3 (Cont.) EVE Procedure Implementing a Variant of the EDT APPEND Command

```
    [TPU$_CONTROLC]:
        eve$$restore_position (saved_mark);
        eve$learn_abort;
        ABORT;
    [OTHERWISE]:        eve$$restore_position (saved_mark);
        eve$learn_abort;
ENDON_ERROR;

remove_range := eve$selection (TRUE);
IF remove_range <> 0
THEN
    saved_mark := MARK (NONE);
    remove_status := eve$test_if_modifiable (GET_INFO (saved_mark, "buffer"));
    IF eve$x_decwindows_active
    THEN
        IF eve$$x_state_array {eve$$k_clipboard}
        THEN
❶           old_string := GET_CLIPBOARD;
            string_range := old_string + str (remove_range);
❷           WRITE_CLIPBOARD ("", new_string);

            IF remove_status
            THEN
                ERASE (remove_range);
                eve$message (EVE$_REMCLIPBOARD);
            ENDIF;
        ELSE
            eve$$edt_append_paste (remove_range, remove_status);
        ENDIF;
    ELSE
        eve$$edt_append_paste (remove_range, remove_status);
    ENDIF;

    POSITION (saved_mark);
    remove_range := 0;
    RETURN (TRUE);
ENDIF;

eve$learn_abort;
RETURN (FALSE);

ENDPROCEDURE;
```

❶ The GET_CLIPBOARD built-in procedure returns a copy of the text stored in the clipboard. Only data of type STRING can be retrieved from the clipboard. Any other data causes VAXTPU to signal an error.

❷ The WRITE_CLIPBOARD built-in procedure stores data in the clipboard. The first parameter allows you to specify the label for this data. However, in this release the clipboard supports only one entry at a time, so you can use any string for the first parameter.

7.7.4 Example of Testing and Returning a Select Range

The code fragment in Example 7–4 shows how a layered application can use GET_GLOBAL_SELECT. This code fragment is a portion of the EVE procedure *eve$selection*. You can find the original version in SYS$EXAMPLES:EVE$CORE.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section 7.7.

The procedure *eve$selection* returns a select range, found range, or global selection for use with EVE commands that operate on the select range.

This example uses the following global variables and procedures from EVE:

- EVE$MESSAGE - A procedure that translates the specified message code into text and displays the text in the message buffer.
- EVE$LEARN_ABORT - A procedure that aborts a learn sequence.
- EVE$X_DECWINDOWS_ACTIVE - A Boolean global variable that is true if VAXTPU is using the DECwindows screen manager. If VAXTPU is not using the DECwindows screen manager, the DECwindows features are not available.

Example 7–4 EVE Procedure Returning a Select Range

```
PROCEDURE eve$selection (
                         do_messages;           ! Display error messages?
                         found_range_arg,       ! Use found range? (D=TRUE).
                         global_arg,            ! Use global select? (D=FALSE).
                         null_range_arg,        ! Extend null ranges? (D=TRUE).
                         cancel_arg)            ! Cancel selection? (D=TRUE).

! Return Values:         range           The selected range.
!                        0               There was no select range.
!                        NONE            There was a null range and
!                                         null_range_arg is FALSE.
!                        string          Text of the global selection
!                                         if "global_arg" is TRUE.

LOCAL   possible_selection,
        use_found_range,
        use_global,
        extend_null_range,
        cancel_range;

ON_ERROR
    [TPU$_SELRANGEZERO]:
    [TPU$_GBLSELOWNER]:
        eve$message (EVE$_NOSELECT);
        eve$learn_abort;
        RETURN (FALSE);
    [OTHERWISE]:
ENDON_ERROR;
```

(continued on next page)

Example 7–4 (Cont.) EVE Procedure Returning a Select Range

```
                                ! The procedure first tests whether it
                                ! has received a parameter directing
                                ! it to return a found range or global
                                ! selection if no select range has been
                                ! created by the user.

IF GET_INFO (found_range_arg, "type") = INTEGER
THEN
    use_found_range := found_range_arg;
ELSE
    use_found_range := TRUE;
ENDIF;

IF GET_INFO (global_arg, "type") = INTEGER
THEN
    use_global := global_arg;
ELSE
    use_global := FALSE;
ENDIF;

!  .
!  .
!  .

                                        ! In the code omitted from this example,
                                        ! eve$selection returns the appropriate
                                        ! range if the calling procedure has
                                        ! requested the user's select range
                                        ! or a found range.

!  .
!  .
!  .

                                        ! If there is no found range or select
                                        ! range, the procedure returns
                                        ! the primary global selection
                                        ! if it exists.

IF use_global and eve$x_decwindows_active
THEN
❶   possible_selection := GET_GLOBAL_SELECT (PRIMARY,
                                             "STRING");

    IF GET_INFO (possible_selection, "type") = STRING
    THEN
        RETURN (possible_selection);
    ENDIF;
ENDIF;
  .
  .
  .
RETURN (0);                     ! Indicates failure.

ENDPROCEDURE;
```

❶ DECwindows allows you to designate more than one global selection. The two most common global selections are the primary and secondary

(continued on next page)

Example 7–4 (Cont.) EVE Procedure Returning a Select Range

selections. A global selection can be owned by only one DECwindows application at a time.

The GET_GLOBAL_SELECT built-in returns the data for the requested selection in the requested format. If the requested selection is not currently owned by any application, or if the owner cannot return it in the requested format, then GET_GLOBAL_SELECT returns UNSPECIFIED.

If the selected information contains multiple records, the records are separated by the line feed symbol (ASCII (10)).

7.7.5 Example of Resizing Windows

Example 7–5 shows a procedure *sample_new_screen_size*, which manipulates visible windows when the user makes the screen smaller. It removes visible VAXTPU windows from the screen, starting at the bottom of the VAXTPU screen, until the combined length of the remaining windows is less than or equal to the new, smaller screen size or until there is only one window left. (For more information about the difference between VAXTPU windows and DECwindows windows, see Chapter 6.) If only one window remains, the procedure adjusts the window to fit the screen. If two or more windows remain, the procedure adjusts the current window and the bottom window.

The procedure uses the following variants of the built-in GET_INFO (window_variable):

- GET_INFO (window_variable, "bottom")
- GET_INFO (window_variable, "length")
- GET_INFO (window_variable, "top")

This example uses the following global variables and procedure from EVE:

- EVE$GET_WINDOW - A procedure that returns the window associated with a number. The windows are numbered sequentially, from top to bottom.
- EVE$X_NUMBER_OF_WINDOWS - A global variable that holds the count of the visible windows.
- EVE$$GET_WINDOW_NUMBER - A procedure that returns a number for the current window. EVE associates a value with each window so EVE can save information about specific windows. This procedure is for EVE internal use only. Do not call this procedure in a user-written procedure.

- EVE$$REMOVE_WINDOW - A procedure that removes a window from the screen. This procedure is for EVE internal use only. Do not call this procedure in a user-written procedure.

Example 7–5 Procedure Resizing Windows

```
PROCEDURE sample_new_screen_size

LOCAL  overhead,
       new_screen_length,
       number,
       the_count,
       total_length,
       some_window,
       a_window,
       new_top,
       a_length,
       top_adjust,
       bottom_window,
       bottom_adjust;

overhead := 2;  ! This provides lines for the command window and message
                ! window, assuming each window has a length of 1.

❶ new_screen_length := get_info (SCREEN, "new_length");

number := eve$$get_window_number;       ! This sets "number" to be
                                        ! the number of the current window.

the_count := eve$x_number_of_windows;   ! This sets "the_count" to
                                        ! be the total number of
                                        ! visible windows.

! The following lines determine the combined lengths of all
! user-created  windows visible on the screen, plus the lengths of the
! command window and message window.

total_length := overhead;
the_count := eve$x_number_of_windows;
LOOP
  EXITIF the_count < 1;
  some_window := eve$get_window (the_count);

  ! "Some_window" is the bottom-most window not yet measured.

  total_length := total_length +
❷                 GET_INFO (some_window, "length", WINDOW);

  the_count := the_count - 1;

ENDLOOP;
```

(continued on next page)

Example 7–5 (Cont.) Procedure Resizing Windows

```
        ! The following statements delete windows from the screen, starting
        ! with the bottom-most window, until the sum of the lengths
        ! of all remaining windows is less than or equal to the new screen
        ! length.
the_count := eve$x_number_of_windows;
LOOP
    EXITIF the_count <= 1;
    a_window := eve$get_window (the_count);  ! This statement sets "a_window" to
                                             ! to be the bottom-most
                                             ! window not yet examined
                                             ! in this loop.

❸   IF number > the_count

    THEN
❹       new_top := GET_INFO (a_window, "top", WINDOW);

    ENDIF;

    IF number <> the_count  ! If the current window is still
                            ! above the window to which you're
                            ! presently comparing it
    THEN
        a_length := GET_INFO (a_window, "length", WINDOW);

           ! The following clause prevents the loop from deleting
           ! the bottom window if the new screen length
           ! is greater than or equal to the old screen length.

        IF new_screen_length < total_length

            ! The following statement decreases "total_length" by the length
            ! of the window presently being examined.

        THEN
            total_length := total_length - a_length;

            ! The following statement removes the window
            ! presently being examined.

            eve$$remove_window (the_count);


        ENDIF;

        EXITIF total_length <= new_screen_length;

    ENDIF;

    the_count := the_count - 1; ! Next time through the loop, the window
                                ! being examined will be the window
                                ! just above the window examined this time.
ENDLOOP;

IF eve$x_number_of_windows = 1
THEN
    adjust_window (CURRENT_WINDOW,
                   1 - get_info (CURRENT_WINDOW, "top", WINDOW),
                   new_screen_length - overhead
❺                  -get_info (CURRENT_WINDOW, "bottom", WINDOW));
ELSE
```

(continued on next page)

Example 7–5 (Cont.) Procedure Resizing Windows

```
    ! The following statements adjust the top of the current
    ! window and the bottom of the bottom window, if needed,
    ! to occupy the space left by deleting windows.

    IF new_top <> 0
    THEN
        top_adjust := new_top - GET_INFO (CURRENT_WINDOW,
                                          "top", WINDOW);
        ADJUST_WINDOW (CURRENT_WINDOW, top_adjust, 0);
    ENDIF;

    bottom_window := eve$get_window (eve$x_number_of_windows);
    bottom_adjust := new_screen_length -
                     overhead -
                     GET_INFO (bottom_window,      ! This statement using
                               "bottom", WINDOW);  ! GET_INFO (window, "bottom")
                                                   ! calculates the amount
                                                   ! by which to adjust the
                                                   ! bottom of the bottom
                                                   ! window.

    ADJUST_WINDOW (bottom_window, 0, bottom_adjust);
ENDIF;

ENDPROCEDURE;
```

❶ GET_INFO (SCREEN "new_length") returns the size of the screen after a resize occurs.

❷ GET_INFO (window, "length", WINDOW) returns the length of the window.

❸ *Number* is greater than *the_count* only when the current window is below the window to which you are comparing it.

❹ GET_INFO (window, "top", WINDOW) returns the top line of the window.

❺ GET_INFO (window, "bottom", WINDOW) returns the line number of the last line in the window.

7.7.6 Example of Unmapping Saved Windows

Example 7–6 shows a procedure *sample_save_window_info_and_unmap*, which saves information about all visible VAXTPU windows in the array *window_array* and then unmaps all visible VAXTPU windows. The windows can be reconstructed later using the information in *window_array*.

The procedure uses the following variants of the built-in GET_INFO (window_variable):

- GET_INFO (window_variable, "width")
- GET_INFO (window_variable, "key_map_list")
- GET_INFO (window_variable, "scroll_bar", VERTICAL)

- GET_INFO (window_variable, "scroll_bar_auto_thumb", VERTICAL)

Note: DIGITAL does not guarantee that this example will work successfully with future versions of EVE.

Example 7–6 EVE Procedure Unmapping Saved Windows

```
PROCEDURE sample_save_window_info_and_unmap (; window_array)

LOCAL   the_count,
        the_window,
        saved_buffer,
        the_row,
        temp;

ON_ERROR
    [TPU$_CONTROLC]:
        IF GET_INFO (saved_buffer, "type") = BUFFER
        THEN
            eve$message (EVE$_REBLDWINDOWS);
            eve$setup_windows (saved_buffer);
            eve$message (EVE$_WINDOWSREBLT);
            UPDATE (ALL);
        ENDIF;
        eve$learn_abort;
        ABORT;
    [OTHERWISE]:
ENDON_ERROR;

the_count := 0;
the_window := GET_INFO (WINDOWS, "first");
LOOP
    EXITIF the_window = 0;
    IF GET_INFO (the_window, "buffer") <> 0
    THEN
        the_count := the_count + 1;
    ENDIF;
    the_window := GET_INFO (WINDOWS, "next");
ENDLOOP;
window_array := CREATE_ARRAY (the_count + 1, 0);
window_array {0} := eve$x_number_of_windows;
the_window := eve$main_window;
IF GET_INFO (the_window, "type") = WINDOW
THEN
    saved_buffer := GET_INFO (the_window, "buffer");
ENDIF;
```

(continued on next page)

Example 7–6 (Cont.) EVE Procedure Unmapping Saved Windows

```
LOOP
    EXITIF the_count = 0;
    the_window := CURRENT_WINDOW;
    EXITIF the_window = 0;
    temp := CREATE_ARRAY (29);
    temp {1} := the_window;
    temp {2} := GET_INFO (the_window, "buffer");
    temp {3} := GET_INFO (the_window, "top", WINDOW);
    temp {4} := GET_INFO (the_window, "length", WINDOW);
    temp {8} := get_info (the_window, "status_line");
    IF temp {8} <> 0
    THEN
        temp {5} := ON;
    ELSE
        temp {5} := OFF;
    ENDIF;
    POSITION (the_window);
    temp {6} := MARK (FREE_CURSOR);
    the_row := GET_INFO (the_window, "current_row");
    IF the_row = 0
    THEN
        the_row := temp {3};
    ENDIF;
    temp {7} := the_row + 1 - temp {3};
    temp {9} := GET_INFO (the_window, "width");     ! This statement uses
                                                    ! GET_INFO (window, "width").

    temp {10} := GET_INFO (the_window, "scroll_top");
    temp {11} := GET_INFO (the_window, "scroll_bottom");
    temp {12} := GET_INFO (the_window, "scroll_amount");
    temp {13} := GET_INFO (the_window, "text");
    temp {14} := GET_INFO (the_window, "blink_video");
    temp {15} := GET_INFO (the_window, "blink_status");
    temp {16} := GET_INFO (the_window, "bold_video");
    temp {17} := GET_INFO (the_window, "bold_status");
    temp {18} := GET_INFO (the_window, "reverse_video");
    temp {19} := GET_INFO (the_window, "reverse_status");
    temp {20} := GET_INFO (the_window, "underline_video");
    temp {21} := GET_INFO (the_window, "underline_status");
    temp {22} := GET_INFO (the_window, "special_graphics_status");
    IF GET_INFO (the_window, "pad")
    THEN
        temp {23} := ON;
    ELSE
        temp {23} := OFF;
    ENDIF;
    temp {24} := GET_INFO (the_window,
                           "shift_amount");
    temp {25} := GET_INFO (the_window,      ! This statement uses
                           "key_map_list"); ! GET_INFO (window, "key_map_list").
```

(continued on next page)

Example 7–6 (Cont.) EVE Procedure Unmapping Saved Windows

```
    IF GET_INFO (SCREEN, "decwindows")
    THEN
        temp {26} := (GET_INFO (the_window,    ! This statement uses
                                "scroll_bar", ! GET_INFO (window, "scroll_bar").
                                VERTICAL) <> 0);
                                                    ! If the vertical
                                                    ! scroll bar is
                                                    ! on, save the
                                                    ! information.

        IF temp {26}
        THEN
            temp {27} := GET_INFO (the_window,              ! This statement uses
                                   "scroll_bar_auto_thumb", ! the GET_INFO
                                   VERTICAL);               ! ("scroll_bar_auto_thumb)
                                                            ! built-in.
        ELSE
            temp {27} := FALSE;
        ENDIF;
        temp {28} := (GET_INFO (the_window, "scroll_bar", HORIZONTAL) <> 0);
        IF temp {28}
        THEN
            temp {29} := GET_INFO (the_window, "scroll_bar_auto_thumb",
                                   HORIZONTAL);
        ELSE
            temp {29} := FALSE;
        ENDIF;
    ELSE
        temp {26} := FALSE;
        temp {27} := FALSE;
        temp {28} := FALSE;
        temp {29} := FALSE;
    ENDIF;
    window_array {the_count} := temp;
    UNMAP (the_window);
    the_count := the_count - 1;
ENDLOOP;
eve$x_number_of_windows := 0;

ENDPROCEDURE;
```

7.7.7 Example of Mapping Saved Windows

Example 7–7 shows the procedure *sample_map_saved_windows*, which maps windows whose descriptions have been saved previously. *Sample_map_saved_windows* is passed the array *window_array* containing information about windows that have previously been saved and then unmapped. You can see an example of how such an array is created in Example 7–6. The procedure maps the windows to buffers, giving the windows the same characteristics they had before they were unmapped.

The procedure includes the following built-ins:

- SET (SCROLL_BAR)

- SET (SCROLL_BAR_AUTO_THUMB)
- SET (WIDGET)
- SET (WIDGET_CALLBACK)

Note: DIGITAL does not guarantee that this example will work successfully with future versions of EVE.

Example 7–7 Procedure Mapping Saved Windows

```
PROCEDURE sample_map_saved_windows (window_array)

LOCAL   temp,
        the_length,
        length_remaining,
        the_top,
        the_count,
        scroll_bar_widget,
        screen_length;

ON_ERROR
    [TPU$_CONTROLC]:
        eve$message (EVE$_RESETUPWINDOWS);
        eve$setup_windows (window_array);
        UPDATE (ALL);
        eve$learn_abort;
        ABORT;
    [OTHERWISE]:
endon_error;

screen_length := eve$get_screen_height;

eve$$unmap_all_windows;

        eve$x_number_of_windows := window_array {0};
        the_count := 1;
        LOOP
            EXITIF the_count > GET_INFO (window_array, "high_index");
            temp := window_array {the_count};
            eve$$map_window (temp {1}, temp {2}, temp {3}, temp {4}, temp {5},
                             temp {6}, temp {7});
            IF temp {5} = ON
            THEN
                SET (STATUS_LINE, temp {1}, NONE, temp {8});
                IF temp {15}
                THEN
                    SET (STATUS_LINE, temp {1}, BLINK, temp {8});
                ENDIF;
                IF temp {17}
                THEN
                    SET (STATUS_LINE, temp {1}, BOLD, temp {8});
                ENDIF;
                IF temp {19}
                THEN
                    SET (STATUS_LINE, temp {1}, REVERSE, temp {8});
                ENDIF;
                IF temp {21}
                THEN
                    SET (STATUS_LINE, temp {1}, UNDERLINE, temp {8});
                ENDIF;
```

(continued on next page)

Example 7–7 (Cont.) Procedure Mapping Saved Windows

```
            IF temp {22}
            THEN
                SET (STATUS_LINE, temp {1}, SPECIAL_GRAPHICS, temp {8});
            ENDIF;
        ENDIF;
        SET (WIDTH, temp {1}, temp {9});
        SET (TEXT, temp {1}, temp {13});
        IF temp {14}
        THEN
            SET (VIDEO, temp {1}, BLINK);
        ENDIF;
        IF temp {16}
        THEN
            SET (VIDEO, temp {1}, BOLD);
        ENDIF;
        IF temp {18}
        THEN
            SET (VIDEO, temp {1}, REVERSE);
        ENDIF;
        IF temp {20}
        THEN
            SET (VIDEO, temp {1}, UNDERLINE);
        ENDIF;
        SET (PAD, temp {1}, temp {23});
        SHIFT (temp {1}, temp {24});
        the_count := the_count + 1;
    ENDLOOP;
    IF GET_INFO (temp {25}, "type") = STRING
    THEN
        SET (KEY_MAP_LIST, temp {25}, temp {1});
    ENDIF;
    IF GET_INFO (SCREEN, "decwindows")
    THEN
        IF temp {26}
        THEN

            scroll_bar_widget := SET (SCROLL_BAR,     ! This statement
                                      temp {1},       ! uses the
                                      VERTICAL, ON);  ! SET (SCROLL_BAR)
                                                      ! built-in.


            SET (WIDGET_CALLBACK,           ! This statement uses the
                 scroll_bar_widget,         ! SET (WIDGET_CALLBACKS)
                 "eve$scroll_dispatch",     ! built-in.
                 'v');

            SET (WIDGET,                            ! This statement uses
                 scroll_bar_widget,                 ! the SET (WIDGET)
                 eve$$scroll_bar_callbacks);        ! built-in.

            eve$$scroll_bar_window (scroll_bar_widget) := temp {1};
            IF temp {27}

            THEN

                SET (SCROLL_BAR_AUTO_THUMB,    ! This statement uses the
                     temp {1}, VERTICAL, ON);  ! SET (SCROLL_BAR_AUTO_THUMB)
                                               ! built-in.

            ENDIF;
```

(continued on next page)

Example 7–7 (Cont.) Procedure Mapping Saved Windows

```
            ENDIF;
            IF temp {28}
            THEN
                scroll_bar_widget := SET (SCROLL_BAR, temp {1}, HORIZONTAL,
                                          ON);
                SET (WIDGET_CALLBACK, scroll_bar_widget, "eve$scroll_dispatch",
                     'h');
                SET (WIDGET, scroll_bar_widget, eve$$scroll_bar_callbacks);
                eve$$scroll_bar_window (scroll_bar_widget) := temp {1};
                IF temp {29}
                THEN
                    SET (SCROLL_BAR_AUTO_THUMB, temp {1}, HORIZONTAL, ON);
                ENDIF;
            ENDIF;
        ENDIF;
        UPDATE (ALL);
        the_count := 1;
        LOOP
            EXITIF the_count > GET_INFO (window_array, "high_index");
            temp := window_array {the_count};
            SET (SCROLLING, temp {1}, ON, temp {10}, temp {11}, temp {12});
            the_count := the_count + 1;
        ENDLOOP;
        SET (PROMPT_AREA, screen_length - 1, 1, REVERSE);
ENDPROCEDURE;
```

7.7.8 Handling Callbacks from a Scroll Bar Widget

Example 7–8 shows one of the ways an application can use the statements POSITION (integer) and SET (WIDGET). The procedure is a portion of the EVE procedure *eve$scroll_dispatch*. You can find the original version in SYS$EXAMPLES:EVE$DECWINDOWS.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section 7.7.

The procedure *eve$scroll_dispatch* is the callback routine handling callbacks from scroll bar widgets. The portion of the procedure shown here determines where to position the editing point based on how the user has changed the scroll bar slider. The procedure fetches the position of the slider with the built-in GET_INFO (widget_variable, "widget_info") and positions the editing point to the line in the buffer equivalent to the slider's position in the scroll bar. Finally, the procedure updates the scroll bar's resource values. For more information about the resource names used with the scroll bar widget, see the *VMS DECwindows Toolkit Routines Reference Manual*.

EVE uses the following constants in this procedure:

- EVEDWTC_NINC - A constant for the string "inc". This is the resource name for the amount that the scroll bar slider position is to be incremented or decremented when a scroll bar button is pressed.

- EVEDWTC_NPAGE_INC - A constant for the string "pageInc". This is the resource name for the amount that the scroll bar slider position is to be incremented or decremented when a click occurs within the scroll bar above or below the slider.
- EVEDWTC_NMAX_VALUE - A constant for the string "maxValue". This is the resource name for the maximum value of the scroll bar slider position.
- EVEDWTC_NMIN_VALUE - A constant for the string "minValue". This is the resource name for the minimum value of the scroll bar slider position.
- EVEDWTC_NVALUE - A constant for the string "value". This is the resource name for the top of the scroll bar slider position.
- EVEDWTC_NSHOWN - A constant for the string "shown". This is the resource name for the size of the slider.
- EVEDWTC_CRVALUE_CHANGE_CALLBACK - A constant for the callback reason code DWT$C_CR_VALUE_CHANGED. This reason code indicates that the user changed the value of the scroll bar slider.
- EVE$K_CLOSURE - A constant for the string "closure", used as an index for the array returned by GET_INFO (WIDGET, "callback_parameters", array).
- EVE$K_REASON_CODE - A constant for the string "reason_code", used as an index for the array returned by GET_INFO (WIDGET, "callback_parameters", array).
- EVE$K_WIDGET - A constant for the string "widget", used as an index for the array returned by GET_INFO (WIDGET, "callback_parameters", array).

Example 7–8 EVE Procedure Handling Callbacks from a Scroll Bar Widget

```
PROCEDURE eve$scroll_dispatch
LOCAL   status,
        widget_called,
        widget_tag,
        widget_reason,
        scroll_bar_values,
        linenum,
        temp_array,
            .
            .
            .;

ON_ERROR
    [TPU$_CONTROLC]:
        eve$learn_abort;
        ABORT;
ENDON_ERROR

❶ status := GET_INFO (WIDGET, "callback_parameters", temp_array);
```

(continued on next page)

Example 7–8 (Cont.) EVE Procedure Handling Callbacks from a Scroll Bar Widget

```
widget_called := temp_array {eve$k_widget};
widget_tag := temp_array {eve$k_closure};
widget_reason := temp_array {eve$k_reason_code};

POSITION (eve$$scroll_bar_window {widget_called});
   .
   .
   .

scroll_bar_values := CREATE_ARRAY;
scroll_bar_values {eve$dwt$c_ninc} := 0;
scroll_bar_values {eve$dwt$c_npage_inc} := 0;
scroll_bar_values {eve$dwt$c_nmax_value} := 0;
scroll_bar_values {eve$dwt$c_nmin_value} := 0;
scroll_bar_values {eve$dwt$c_nvalue} := 0;
scroll_bar_values {eve$dwt$c_nshown} := 0;

❷ status := GET_INFO (widget_called, "widget_info", scroll_bar_values);

! The deleted statements scroll the window as dictated
! by the callback reason.

   .
   .
   .

CASE widget_reason
   .
   .
   .

        [eve$dwt$c_crvalue_change_callback]:

           IF (scroll_bar_values {eve$dwt$c_nvalue} =
               scroll_bar_values {eve$dwt$c_nmin_value})
           THEN
               POSITION (beginning_of (current_buffer));
           ELSE
❸              POSITION (scroll_bar_values {eve$dwt$c_nvalue});
            ENDIF;
                  .
                  .
                  .

scroll_bar_values {eve$dwt$c_ninc} := 1;
scroll_bar_values {eve$dwt$c_npage_inc} := scroll_bar_values {eve$dwt$c_nshown}
                                           - 1;

❹ SET (WIDGET, widget_called, scroll_bar_values);

!      .
!      .
!      .

ENDPROCEDURE;
```

❶ GET_INFO (WIDGET, "callback_parameters", array) returns an array containing the values for the current callback. The array elements are indexed by the strings "widget", "closure", and "reason_code".

(continued on next page)

Example 7–8 (Cont.) EVE Procedure Handling Callbacks from a Scroll Bar Widget

referencing the widget that is calling back, the widget's closure value, and the reason code for the callback.

❷ GET_INFO (widget, "widget_info", array) allows you to fetch information from a widget. The array parameter is indexed by the resource names associated with the specified widget. Note that resource names are case sensitive. Note, too, that the set of supported resources varies from one widget type to another. When you use GET_INFO (widget, "widget_info", array), VAXTPU queries the widget for the requested information and puts the returned information in the array elements. Any previous values in the array are lost.

❸ POSITION (integer) allows you to move the editing point to the record specified by the parameter *integer*. VAXTPU interprets this parameter as a record number.

❹ SET (WIDGET, widget, array) allows you to set resource values for the specified widget. The array parameter is indexed by the resource names associated with the specified widget. Note that resource names are case sensitive. Note, too, that the set of supported resources varies from one widget type to another.

7.7.9 Example of Implementing the COPY SELECTION Operation

Example 7–9 shows one of the ways an application can use the READ_GLOBAL_SELECT built-in. The procedure is a modified version of the EVE procedure *eve$stuff_global_selection*. You can find the original version in SYS$EXAMPLES:EVE$MOUSE.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section 7.7.

The procedure performs the following tasks:

- Saves the location of the editing point and the buffer's current mode.
- Checks that the DECwindows version of EVE is enabled and that EVE does not have input focus.
- Obtains the location of the pointer cursor and positions the editing point at that location.
- Sets the text insertion mode to the insert setting.
- Reads the string-formatted contents of the primary global selection. (In this context, the parameter "*STRING*" means that the calling application is asking the application that owns the global selection for the string-formatted information in the specified global selection.)
- Restores the editing point location and text insertion mode to their previous values.

EVE binds this procedure to the MB3 key to implement the DECwindows COPY SELECTION operation from another application to EVE.

Example 7–9 EVE Procedure Implementing the COPY SELECTION Operation

```
PROCEDURE eve$stuff_global_selection

LOCAL   saved_position,
        saved_mode,
        this_buffer,
        the_window,
        the_column,
        the_row;

ON_ERROR
    [TPU$_CONTROLC]:
        IF saved_mode = OVERSTRIKE
        THEN
            SET (saved_mode, this_buffer);
        ENDIF;
        eve$$restore_position (saved_position);
        eve$learn_abort;
        ABORT;
    [OTHERWISE]:
        IF saved_mode = OVERSTRIKE
        THEN
            SET (saved_mode, this_buffer);
        ENDIF;
        eve$$restore_position (saved_position);
ENDON_ERROR;

this_buffer := current_buffer;
saved_position := MARK (FREE_CURSOR);
saved_mode := GET_INFO (this_buffer, "mode");

IF eve$x_decwindows_active
THEN
    IF not GET_INFO (SCREEN, "input_focus")
    THEN
        IF LOCATE_MOUSE (the_window,              ! This statement uses
                         the_column, the_row)     ! the LOCATE_MOUSE built-in.
        THEN
            IF the_row <> 0
                IF the_window <> eve$choice_window
                THEN
                    POSITION (MOUSE);
                    SET (INSERT, this_buffer);

                    READ_GLOBAL_SELECT (PRIMARY, "STRING");  ! This statement
                                                             ! using READ_GLOBAL_SELECT
                                                             ! reads the string-
                                                             ! formatted contents
                                                             ! of the primary
                                                             ! global selection.
```

(continued on next page)

Example 7–9 (Cont.) EVE Procedure Implementing the COPY SELECTION Operation

```
                    eve$$restore_position (saved_position);
                    SET (saved_mode, this_buffer);
                    UPDATE (CURRENT_WINDOW);
                    RETURN (TRUE);
                ENDIF;
            ENDIF;
        ENDIF;
    ENDIF;
ENDIF;

RETURN (FALSE);

ENDPROCEDURE;
```

7.7.10 Example of Reactivating a Select Range

Example 7–10 shows one of the ways an application can use the SET (GLOBAL_SELECT) built-in. The procedure is a modified version of the EVE procedure *eve$restore_primary_selection*. You can find the original version in SYS$EXAMPLES:EVE$MOUSE.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section 7.7.

The procedure *eve$restore_primary_selection* reactivates EVE's select range when EVE regains input focus.

Example 7–10 EVE Procedure Reactivating a Select Range

```
PROCEDURE eve$restore_primary_selection

LOCAL   saved_position;

ON_ERROR
    [TPU$_CONTROLC]:
        eve$$restore_position (saved_position);
        eve$learn_abort;
        ABORT;
    [OTHERWISE]:
        eve$$restore_position (saved_position);
ENDON_ERROR;

IF NOT eve$x_decwindows_active
THEN
    RETURN (FALSE);
ENDIF;

saved_position := MARK (FREE_CURSOR);
```

(continued on next page)

Example 7–10 (Cont.) EVE Procedure Reactivating a Select Range

```
IF GET_INFO (eve$$x_save_select_array, "type") = ARRAY
THEN
    CASE eve$$x_save_select_array {"type"}
        [RANGE]:
            eve$select_a_range (eve$$x_save_select_array {"start"},
                                eve$$x_save_select_array {"end"});
            eve$$x_state_array {eve$$k_select_all_active} :=
                                                    eve$$x_save_select_array
                                                        {"select_all"};
            POSITION (eve$$x_save_select_array {"current"});
            eve$start_pending_delete;
        [MARKER]:
            POSITION (eve$$x_save_select_array {"start"});
            eve$x_select_position := select (eve$x_highlighting);
            POSITION (eve$$x_save_select_array {"end"});
            eve$start_pending_delete;
        [OTHERWISE]:
            RETURN (FALSE);
    ENDCASE;

    eve$$restore_position (saved_position);
    eve$$found_post_filter;             ! This is necessary if the
                                        ! cursor is outside the selection.

    eve$$x_save_select_array {"type"} := 0;
    UPDATE (current_window);
    IF eve$x_decwindows_active
    THEN

        SET (GLOBAL_SELECT, SCREEN, PRIMARY);  ! This statement using
                                               ! SET (GLOBAL_SELECT)
                                               ! requests ownership of
                                               ! the primary global selection.
    ENDIF;
    RETURN (TRUE);
ENDIF;

RETURN (FALSE);

ENDPROCEDURE;
```

7.7.11 Example of Implementing the DECwindows COPY SELECTION Operation from EVE to Another Application

Example 7–11 shows one of the ways a layered application can use the WRITE_GLOBAL_SELECT built-in. The procedure is a modified version of the EVE procedure *eve$write_global_select*. You can find the original version in SYS$EXAMPLES:EVE$MOUSE.TPU. For more information about using the files in SYS$EXAMPLES as examples, see Section 7.7.

The procedure implements the DECwindows COPY SELECTION operation from EVE to another application. The procedure determines what property of the primary global selection is being requested, obtains the value of the appropriate property using a GET_INFO statement or an EVE routine, and sends the information to the requesting application.

Example 7–11 EVE Procedure Implementing COPY SELECTION

```
PROCEDURE eve$write_global_select          ! EVE uses this routine
                                           ! to respond to requests
                                           ! for information about
                                           ! selections.

LOCAL   saved_position,
        the_data,
        temp_array,
        total_lines,
        the_line,
        status,
        eob_flag,
        percent;

ON_ERROR
    [OTHERWISE]:
        eve$$restore_position (saved_position);
ENDON_ERROR;

saved_position := MARK (FREE_CURSOR);

IF NOT eve$x_decwindows_active

THEN
    RETURN (FALSE);
ENDIF;

the_data := "";
temp_array := GET_INFO (SCREEN, "event", GLOBAL_SELECT);

    ! Finds out which global selection and which property
    ! of the global selection are the subject of the
    ! information request.

CASE temp_array {2}     ! Determines the property requested by other application.

    ["STRING", "TEXT"]: ! If one of these strings is requested, the
                        ! procedure sends the text in the global
                        ! selection to the requesting application.

        CASE temp_array {1}  ! Checks which global selection was specified.
            [PRIMARY]:

                IF eve$x_select_position <> 0
                THEN
                    POSITION (GET_INFO (eve$x_select_position, "buffer"));
```

(continued on next page)

Example 7–11 (Cont.) EVE Procedure Implementing COPY SELECTION

```
                    IF GET_INFO (eve$x_select_position, "type") = RANGE
                    THEN
                        the_data := STR (eve$x_select_position);
                    ELSE
                        IF GET_INFO (eve$x_select_position, "type") = MARKER
                        THEN
                            the_data := STR (eve$select_a_range (eve$x_select_position,
                                                                 MARK (FREE_CURSOR)));
                        ELSE
                            the_data := NONE;
                        ENDIF;
                    ENDIF;
                    eve$$restore_position (saved_position);
                ENDIF;
            [OTHERWISE]:
                the_data := NONE;
        ENDCASE;
    [OTHERWISE]:
        the_data := NONE;       ! The procedure does not send data if
                                ! the requesting application has asked
                                ! for something other than the text,
                                ! the file name, or the line number.
ENDCASE;

WRITE_GLOBAL_SELECT (the_data);         ! This statement sends the
                                        ! requested information to
                                        ! the requesting application.

ENDPROCEDURE;
```

8 CDA Converter Architecture

The CDA Converter Architecture defines a methodology to simplify the conversion of compound documents. Compound documents contain integrated components such as proportionally spaced text, synthetic graphics, and scanned or natural images. For more information on compound documents or the Compound Document Architecture, see the *VMS Compound Document Architecture Manual.*

The CDA Converter Architecture is implemented through the following applications:

- The CDA Converter
- The Character Cell Viewer
- The supported front and back ends

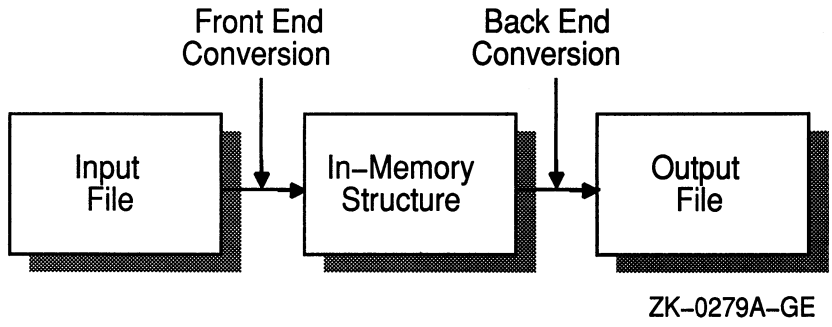
The following sections discuss each of these applications in more detail. Note that a DDIF Viewer and a PostScript are supported on VMS DECwindows systems. These viewers are described in the *VMS DECwindows User's Guide.*

8.1 CDA Converter

The CDA Converter is an integral part of the Compound Document Architecture. It enables you to translate your compound document files to and from various file-encoding formats. The CDA Converter can be viewed as a "black box" that reads in an input file of the specified file-encoding format and converts it to an output file of the specified file-encoding format.

To accomplish this conversion, the CDA Converter uses an in-memory format as the integral step in the conversion process. The converter reads the input file and translates it to a CDA in-memory format, and then translates this in-memory format to the specified output format. In other words, any input file-encoding format that is supported by the CDA Converter can be translated to a CDA in-memory format, and this in-memory format can subsequently be converted to any supported output file-encoding format. Figure 8–1 illustrates these basic stages of document conversion.

Figure 8–1 Stages of Document Conversion



8.1.1 Components of a Converter

From the user's perspective, the converter is a "black box" that reads in the specified input file and converts it to the specified output file. For this single converter to be able to convert the wide variety of supported file-encoding formats, it actually comprises the following four parts:

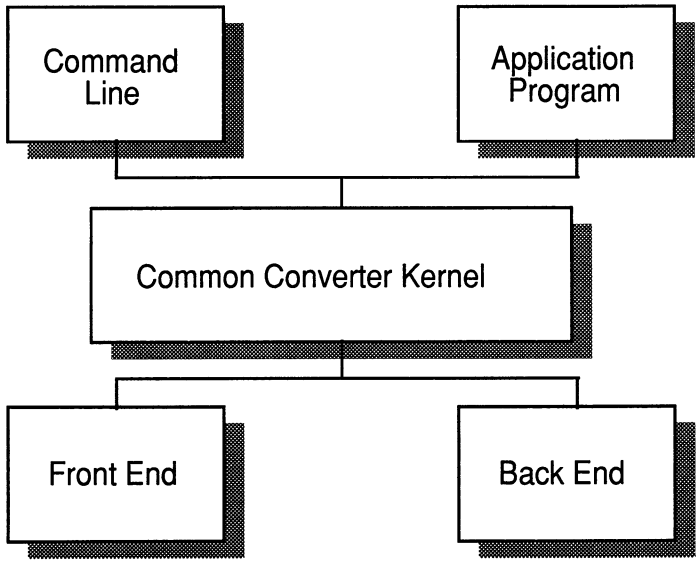
1. An interface (both a command line interface and an interface that is callable from within an application program)
2. The CDA Converter Kernel that performs all the functions that must be completed for each conversion process, regardless of input and output formats
3. A front end that converts a particular input format to the in-memory format
4. A back end that converts the in-memory format to a particular output format

The relationship of the various converter components is shown in Figure 8–2.

When you invoke the converter, you always invoke the converter kernel first. This kernel performs the following functions:

- It performs all of the "generic" conversion functions that must be completed for every document conversion, regardless of input and output formats.
- It invokes the appropriate front end to translate the input file to the CDA in-memory format.
- It invokes the appropriate back end to translate the CDA in-memory format to an output file of the specified format.

Figure 8–2 Converter Components Diagram



The CDA Converter, therefore, actually consists of the CDA Converter Kernel, one front end for each supported input file-encoding format, and one back end for each supported output file-encoding format. The kernel translates the various file formats by calling the appropriate front end and back end to perform the requested conversion.

For example, if you have the CDA Converter Kernel, a DDIF front end, and an Analysis back end, you can invoke the converter to translate a DDIF-encoded input file to an Analysis-encoded output file. The common converter kernel invokes the DDIF front end and the Analysis back end to perform the requested conversion. In general, front ends and back ends are "paired." That is, if a file-encoding format is supported by a front end, it generally is also supported by a back end. However, this is not always the case. For example, the Analysis back end does not have a corresponding front end.

The front ends and back ends that are provided with the operating system are documented later in this chapter. The interfaces to the CDA Converter are as follows:

- A DCL command line interface (CONVERT/DOCUMENT).
- A callable interface (the CONVERT routine) that is accessible from application programs (see the *VMS Compound Document Architecture Manual* for more information).

Each of these interfaces is discussed in the following sections. The Character Cell Viewer is discussed in Section 8.2. The supported input formats are discussed in Section 8.3 and the supported output formats are discussed in Section 8.4.

8.1.2 DCL Command CONVERT/DOCUMENT

The DCL command CONVERT/DOCUMENT invokes the conversion of a revisable format file to another revisable or final form file from the DCL command line. Please note that this command can only be used if you have DECwindows installed on your system. This command has the following format:

```
CONVERT/DOCUMENT[/OPTIONS=filespec]
    input-file[/FORMAT=fmt-name]  output-file[/FORMAT=fmt-name]
```

The /FORMAT qualifier enables you to specify the encoding formats of the input and output files. (DDIF is the default input and output format.) The format keywords for the supported input and output formats are listed in Table 8–1.

Table 8–1 Converter Format Keywords

Input Formats	Output Formats
DDIF	DDIF
TEXT	TEXT
N/A	PS
N/A	ANALYSIS

The /OPTIONS qualifier enables you to specify a file that contains options to be applied during the conversion of the file. Each line of the file specifies a format name that can contain uppercase and lowercase alphabetic characters, digits, dollar signs, and underscores, optionally preceded by spaces and tabs, and terminated by any character other than those listed. Alphabetic case is not significant. The syntax and interpretation of the text that follows the format name are specified by the supplier of the front and back ends for the specified format. Multiple lines that specify the same format are permitted.

8.2 Character Cell Viewer

The Character Cell Viewer is an application that enables you to view compound document files on a character cell terminal or workstation window. This character cell viewer works with the CDA Converter Architecture, so that a file of any input format supported by CDA can be viewed on a character cell terminal.

The Character Cell Viewer works by converting an input file to the in-memory format used by the CDA Converter. This in-memory format is then formatted for output to the screen. In other words, the Character Cell Viewer is a specific instance of the CDA Converter in which the output format is a screen display.

The interfaces to the Character Cell Viewer are as follows:

- A DCL command line interface (VIEW)

- A callable interface (the CONVERT routine) that is accessible from application programs

Each of these interfaces is discussed in the following sections. The supported input formats are discussed in Section 8.3.

8.2.1 DCL Command VIEW

The DCL command VIEW invokes the Character Cell Viewer, which lets you view a compound document file on a character cell terminal or DECwindows display. Note that many of the text display attributes are not processed when displaying the document, because of the limitations of the viewing device. For example, the blink, underlining, and bold attributes are not processed.

The VIEW command has the following format:

```
VIEW input-file[/qualifiers]
```

The input file specifies the name of the file to be viewed. You cannot use wildcard characters in the file specification. The default input file-encoding format is DDIF, and the default file type is DDIF. Valid input file formats are DDIF and TEXT; these input formats are described in more detail in Section 8.3.

The qualifiers that you can specify to the view command are as follows:

- /FORMAT[=format-name]

  Specifies the format of the input file. The default format is DDIF. The appropriate front end must be available in SYS$LIBRARY for the specified **format-name**. The valid formats are DDIF and TEXT.

- /OUTPUT[=output-file-spec]

  Specifies a file that receives the text output. The default is /NOOUTPUT. If an output file specification is not specified, the output file specification defaults to input-file.LIS. If this qualifier is specified, the output of the VIEW command is not displayed on the screen, but is instead written to the specified file. Note that if you specify the /OUTPUT qualifier, you cannot also specify the /PAGE qualifier.

- /PAGE

  Controls the display of output, providing the same effect as the DCL command TYPE/PAGE when used on a non-DECwindows device. The default is /NOPAGE. The /PAGE qualifier has no effect when used with a DECwindows display because the scroll bars provide the same capability. Note that if you specify the /PAGE qualifier, you cannot also specify the /OUTPUT qualifier.

- /OPTIONS=file-spec

  Specifies a file that contains options to be applied during the conversion of the file to the CDA in-memory format. The default file type is DDIF$OPTIONS.

- /SELECT=select-list

Allows the user to tailor the CDA Viewer output. The selection items you can specify are as follows:

[NO]GRAPHICS	Directs the viewer either to mark the location of graphics embedded in the DDIF file being processed by the character cell viewer, or to ignore the graphics.
[NO]IMAGES	Directs the viewer either to mark the location of the images embedded in the DDIF file being processed by the character cell viewer, or to ignore the images.
[NO]TEXT	Directs the viewer either to process the text contained in the DDIF file being processed, or to ignore the text.
ALL	Directs the viewer to process all information contained in the DDIF file being processed.
[NO]SOFT_DIRECTIVES	Directs the viewer either to process or ignore soft directives in the DDIF file being processed in order to format output. Soft directives specify such formatting commands as new line, new page, and tab.
[NO]AUTO_WRAP	Directs the viewer to perform word wrapping of any text that would exceed the right margin. NOAUTO_WRAP allows the text to exceed the margin.
[NO]X_DISPLAY	Directs the viewer to create a DECwindows widget to be used when viewing the file on a workstation display defined by the logical name DECW$DISPLAY. NOX_DISPLAY, the default, invokes the character cell viewer. Note that X_DISPLAY cannot be specified if the /OUTPUT qualifier is also specified.

The default format is

```
/SELECT = (GRAPHICS, IMAGES, TEXT, SOFT_DIRECTIVES,
           AUTO_WRAP, NOX_DISPLAY)
```

Because the Character Cell Viewer is really a specific instance of the CDA Converter (where the output format is screen output), the CONVERT routine is used to invoke the Character Cell Viewer as well as the CDA Converter. For more information on the CONVERT routine, see the *VMS Compound Document Architecture Manual*.

8.3 Input Formats

The CDA Converter Architecture works by supplying a common converter kernel and front and back ends to support the various input and output formats. The following sections describe each supported front end, the data mapping between that input format and the in-memory format, any data loss that might occur during the conversion, and any other information specific to that front end.

8.3.1 DDIF Front End

The DDIF front end reads a file encoded in DDIF format and converts the information in the file to the CDA in-memory structure. The following information is specific to the DDIF front end:

- Data Mapping. Because the input file format is DDIF, the information in the file maps directly to the CDA in-memory structure.
- Data Loss. The DDIF front end does not lose any data when converting a DDIF input file to the CDA in-memory structure. Again, this is because the input document type and the in-memory structure type are both DDIF.
- External File References. When the DDIF front end encounters an external file reference that is specified in the document header of your DDIF input file, it passes the reference through to the CDA Converter Kernel.
- Document Syntax Errors. If a document syntax error is encountered in the DDIF front end, that represents a fatal input processing error. The only way that this can occur is if the input document is invalid. If the DDIF front end does encounter a document syntax error, the conversion process is stopped and no further input processing is performed.

8.3.2 Text Front End

The Text front end reads a standard text (DEC multinational) file and converts the information in the file to the CDA in-memory structure. The following information is specific to the Text front end:

- Data Mapping. When you invoke the converter for a Text input file, all of the text in the input file is mapped to DDIF text content. Line breaks and form feeds are mapped to DDIF directives.
- Data Loss. The Text front end does not lose any data when converting a Text input file to the CDA in-memory structure. This is because no structure information is contained in a text file.
- External File References. Text files do not contain external file references. Therefore, the Text front end does not evaluate external file references.
- Document Syntax Errors. Because text files do not have any syntax defined, syntax errors cannot be encountered by the Text front end.

8.4 Output Formats

The following sections describe each back end supported by the CDA Converter Architecture, the data mapping between the in-memory format and the particular output format, any data loss that might occur during the conversion, and any other information specific to that back end.

8.4.1 DDIF Back End

The DDIF back end takes the CDA in-memory structure that has been converted from some input format, converts it to a DDIF output format, and writes the information to the specified DDIF output file. The following information is specific to the DDIF back end:

- Data Mapping. When you invoke the converter with the DDIF back end, the data mapping between the information in the CDA in-memory structure and the converted output file is one-to-one. This is because the in-memory structure type and the output document type are both DDIF.
- Data Loss. The DDIF back end does not lose any data when converting a CDA in-memory structure to a DDIF output file. Again, this is because the in-memory structure type and the output document type are both DDIF.

8.4.2 Text Back End

The Text back end takes the CDA in-memory structure that has been converted from some input format, converts only the text content of the file, and writes the information to the specified text output file. The following information is specific to the Text Back End:

- Data Mapping. When you invoke the converter for a text output file, all Latin1 text is written to the output text file.
- Data Loss. When the Text back end is converting the in-memory structure to a text output file, all graphics, images, attributes, and formatting information are lost.
- Processing Options. The text back end supports the following options:

ASCII_FALLBACK	This option causes the back end to output text in 7-bit ASCII. The fallback representation of the characters is described in the ASCII ANSI standard.
CONTENT_MESSAGES	This option causes the back end to put a message in the output file each time a nontext element is encountered in the in-memory CDA structures.

8.4.3 PostScript Back End

The PostScript back end takes the CDA in-memory structure that has been converted from some input format, converts the content of the file, and writes the information to the specified text output file. The following information is specific to the PostScript Back End:

- Data Mapping. When you invoke the converter for a PostScript output file, all document content is written to the output file.
- Data Loss. When converting the in-memory structure to a PostScript output file, all document content is converted.

- Processing Options.

  The PostScript back end supports various processing options. The keyword is separated from its assigned value by one or more spaces or tabs. Note that, for all of the measurement options, the default unit of measure is inches. Other supported units of measure are points, centimeters (cm), and millimeters (mm).

  The processing options are described as follows:

  — Paper Size Processing Option.

  The PAPER_SIZE *paper-size* option lets you specify the size of the paper to be used when formatting the resulting PostScript output file. Valid values for *paper-size* are as follows:

Keyword	Size
A0	841 x 1189 millimeters
A1	594 x 841 millimeters
A2	420 x 594 millimeters
A3	297 x 420 millimeters
A4	210 x 297 millimeters
A	8.5 x 11 inches
B	11 x 17 inches
C	17 x 22 inches
D	22 x 34 inches
E	34 x 44 inches
LEDGER	11 x 17 inches
LEGAL	8.5 x 14 inches
LETTER	8.5 x 11 inches
LP	13.7 x 11 inches
VT	8 x 5 inches

  The A paper size (8.5 x 11 inches) is the default.

  — Paper Height Processing Option.

  The PAPER_HEIGHT *paper-height* processing option, in combination with the PAPER_WIDTH processing option, lets you specify a paper size other than one of the predefined values provided. The default paper height is 11 inches.

  — Paper Width Processing Option.

  The PAPER_WIDTH *paper-width* processing option, in combination with the PAPER_HEIGHT processing option, lets you specify a paper size other than one of the predefined sizes provided. The default paper width is 8.5 inches.

— Top Margin Processing Option.

The PAPER_TOP_MARGIN *top-margin* processing option lets you select the width of the margin provided at the top of the page. The default value is

— Bottom Margin Processing Option

The PAPER_BOTTOM_MARGIN *bottom-margin* processing option lets you select the width of the margin provided at the bottom of the page. The default value is .25 inches.

— Left Margin Processing Option

The PAPER_LEFT_MARGIN *left-margin* processing option lets you select the width of the margin provided on the left-hand side of the page. The default value is .25 inches.

— Right Margin Processing Option.

The PAPER_RIGHT_MARGIN *right-margin* processing option lets you select the width of the margin provided on the right-hand side of the page. The default value is .25 inches.

— Paper Orientation Processing Option.

The PAPER_ORIENTATION *orientation* processing option lets you select the paper orientation to be used in the output PostScript file. The valid values for the *orientation* argument are as follows:

Keyword	Meaning
PORTRAIT	The page is oriented so that the larger dimension is parallel to the vertical axis.
LANDSCAPE	The page is oriented so that the larger dimension is parallel to the horizontal axis.

The default is PORTRAIT.

— Eight Bit Output Processing Option.

The EIGHT_BIT_OUTPUT *eight-bit-output-state* processing option lets you select whether or not the PostScript back end should use 8-bit output. You can specify a value of either ON or OFF for the *eight-bit-output-state* argument. The default is ON.

— Output Buffer Size Processing Option.

The OUTPUT_BUFFER_SIZE *output-buffer-size* processing option lets you select the size of the output buffer. The value you specify must be within the following range:

$$64 \leq output - buffer - size \leq 256$$

The default is 132.

— Soft Directives Processing Option.

The SOFT_DIRECTIVES *soft-directives-state* processing option lets you select whether or not the PostScript back end processes soft directives in the DDIF file in order to format output. (Soft directives specify such formatting commands as new line, new page, and tab.) You can specify a value of either ON or OFF for the *soft-directive-state* argument. The default is ON.

— Word Wrap Processing Option.

The WORD_WRAP *word-wrap-state* processing option lets you specify whether or not the PostScript back end performs word wrapping of any text that would exceed the right margin. You can specify a value of either ON or OFF for the *word-wrap-state* argument. The default is ON. If you specify OFF, the PostScript back end allows text to exceed the right margin.

— Page Wrap Processing Option.

The PAGE_WRAP *page-wrap-state* processing option lets you specify whether or not the PostScript back end performs page wrapping of any text that would exceed the bottom margin. You can specify a value of either ON or OFF for the *page-wrap-state* argument. The default is ON.

— Layout Processing Option.

The LAYOUT *layout-state* processing option lets you specify whether or not the PostScript back end processes the layout specified in the DDIF document. You can specify a value of either ON or OFF for the *layout-state* argument. The default is ON.

8.4.4 Analysis Back End

This back end produces an analysis of the CDA in-memory structure in the form of text output showing the named objects and values stored in the document. This is useful for debugging DDIF application programs.

The Analysis back end supports an /INHERITANCE processing option that specifies that the analysis is shown with attribute inheritance enabled. Inherited attributes are marked by "[default]" in the output.

9 Support for Compound Documents

The term **compound documents** refers to files that may contain a number of integrated components including text, graphics, and scanned images. This chapter specifically describes VMS support for using the text from DECwindows compound documents that are structured in accordance with the Digital Data Interchange Format (DDIF) specification. Refer to the *VMS Compound Document Architecture Manual* for more information about compound documents.

VMS commands and utilities, as well as existing application programs that accept text input, can now use the text content of DECwindows compound documents.

To support the use of DDIF text, VMS RMS has implemented a new RMS file attribute, **stored semantics,** and a DDIF-to-text **RMS extension.** The value of the stored semantics attribute is called the file **tag** and it specifies how file data is to be interpreted. When file data is to be interpreted in accordance with the DDIF specification, the appropriate file tag is DDIF. The use of file tags is limited to disk files on VMS Version 5.1 and later systems.

The DDIF-to-text RMS extension transparently extracts text from DDIF files as variable-length text records that can be accessed through the VMS RMS interface.

The enhancements made to support the reading of text from DDIF files are transparent to the user and to the application programmer. This support requires that all DDIF files in a VMS Version 5.1 environment be tagged with the DDIF file tag. DDIF files created by VMS and VMS layered products are tagged appropriately.

Section 9.1 describes various VMS_file management commands and utilities that display, create, and preserve file tags where appropriate. Section 9.1 also describes the way various VMS commands and utilities respond to DDIF file input. Section 9.2 describes VMS support for DDIF files in heterogeneous computing environments. Section 9.3 describes the changes made to the VMS RMS program interface to support the stored semantics attribute and to control access to the content of DDIF files.

9.1 VMS Commands and Utilities

This section describes the VMS commands and utilities that support tag maintenance by displaying, creating, and preserving the RMS file tags used with DDIF files. It also provides additional information that is relevant to the way selected VMS commands and utilities respond to DDIF file input.

Support for Compound Documents

9.1 VMS Commands and Utilities

The following table lists the VMS commands and utilities that support tag maintenance.

Command/Utility	Tag Maintenance Function
DIRECTORY/FULL	Displays file tag
ANALYZE/RMS_FILE	Displays file tag
SET FILE/SEMANTICS	Creates file tag
VMS MAIL	Preserves file tag†
COPY	Preserves file tag†
BACKUP	Preserves file tag

†See text for exceptions.

Tags are made up of binary values that can be up to 64 bytes long and can be expressed using hexadecimal notation. The hexadecimal value of the DDIF tag, for example, is 2B0C8773010301. VMS permits you to assign mnemonics to tag values so that DCL commands like DIRECTORY/FULL and VMS utilities like FDL and ANALYZE/RMS_FILE display a mnemonic for the DDIF tag instead of the hexadecimal value. The following DCL commands have been included in the system startup command file to assign the mnemonic DDIF to the hexadecimal value for a DDIF tag:

```
$ DEFINE/TABLE=RMS$SEMANTIC_TAGS DDIF 2B0C8773010301
$ DEFINE/TABLE=RMS$SEMANTIC_OBJECTS 2B0C8773010301 DDIF
```

Using the appropriate DEFINE commands, you can assign mnemonics for other tags, including tags used with international program applications.

9.1.1 Displaying RMS File Tags

The DIRECTORY/FULL command and the Analyze/RMS_File Utility now display the RMS file tag for DDIF files.

9.1.1.1 DIRECTORY/FULL

Where applicable, the DIRECTORY/FULL command now provides the value of the stored semantics tag as part of the file information returned to the user. This is the recommended method for quickly determining whether or not a file is tagged. The following display illustrates how the DIRECTORY/FULL command returns the RMS attributes for a DDIF file named X.DDIF:

```
X.DDIF;1                        File ID:  (767,20658,0)
.
.
.
RMS attributes:      Stored semantics: DDIF
.
.
.
```

9.1.1.2 ANALYZE/RMS_FILE

When you use the ANALYZE/RMS_FILE command to analyze a DDIF file, the utility returns the file tag as an RMS file attribute.

```
FILE HEADER
File Spec: USERD$:[TEST]X.DDIF;1
    .
    .
    .
Stored semantics: DDIF.
    .
    .
    .
```

One ANALYZE/RMS_FILE command option is to create an output FDL file that reflects the results of the analysis.

```
$ ANALYZE/RMS_FILE/FDL filespec
```

When you use this option for analyzing a tagged file, the output FDL file includes the file tag as a secondary attribute to the FILE primary attribute. This is illlustrated in the following FDL file excerpt:

```
IDENT "9-JUN-1988 13:27:30 VAX/VMS ANALYZE/RMS_FILE Utility"
    .
    .
    .
SYSTEM
        SOURCE                  VMS
FILE
        ALLOCATION              3
    .
    .
    .
        STORED_SEMANTICS        %X'2B0C8773010301' ! DDIF
    .
    .
    .
```

9.1.2 Creating RMS File Tags

The CDA$CREATE_FILE routine in the Compound Document Architecture toolkit creates and tags DDIF files. However, you may encounter a DDIF file that was created without a file tag or a DDIF file whose file tag was not preserved during file processing.

The DCL command SET FILE provides a new qualifier, /[NO]SEMANTICS, that permits you to tag a DDIF file through the DCL interface for VMS Version 5.1 systems. You can also use the qualifier to change a tag or to remove a tag from a file.

The following command line tags the file X.DDIF as a DDIF file by assigning the appropriate value to the /SEMANTICS qualifier:

```
$ SET FILE X.DDIF/SEMANTICS=DDIF
```

See Section 9.1 for information about how to use logical name tables to assign a mnemonic to a tag.

A subsequent DIRECTORY/FULL command displays the following line as part of the file header:

```
.
.
.
RMS attributes:     Stored semantics: DDIF
.
.
.
```

The next example illustrates how to use the SET FILE command to delete an RMS file tag:

```
$ SET FILE X.DDIF/NOSEMANTICS
```

9.1.3 Preserving RMS File Tags and DDIF Semantics

The COPY command and the VMS Mail Utility preserve RMS file tags and DDIF semantics when you copy or mail a DDIF file on a VMS Version 5.1 system, except for conditions described in Sections 9.2.2, 9.2.3, and 9.2.4.

The Backup Utility always preserves file tags and semantics when you back up a DDIF file to magnetic tape.

9.1.3.1 COPY Command

This section describes the results of using the COPY command with DDIF files for various operations.

When you copy a DDIF file to a disk on a VMS Version 5.1 system using the COPY command, VMS RMS preserves the DDIF tag and the DDIF semantics of the input file in the output file.

When you copy a DDIF file to a nondisk device on a VMS Version 5.1 system using the COPY command, VMS RMS does *not* preserve the DDIF tag or the DDIF semantics of the input file in the output file. Instead, VMS RMS writes the text from the input file to the output file as variable-length records.

When you copy two or more DDIF and text files in any combination to a single output file, the output file takes the characteristics of the first input file, as shown in the following examples:

1 In the first example, the first input file is a text file, so the output file (FOO.TXT) contains variable-length text records from X.TXT, Y.DDIF, and Z.TXT, but does not include the DDIF tag from Y.DDIF.

```
$ COPY X.TXT,Y.DDIF,Z.TXT FOO.TXT
```

2 In the next example, the first input file (A.DDIF) is a DDIF file, so the output file (FOO.DDIF) includes the DDIF tag as well as the DDIF semantics from A.DDIF. The attempt to copy the text input file (Z.TXT) fails because there is no text-to-DDIF RMS extension, but the contents of B.DDIF and C.DDIF are copied to the output file. However, the output file has no practical use because, as a result of the way DDIF files are structured, only the data from the first input file (A.DDIF) is accessible in the output file.

```
$ COPY A.DDIF,B.DDIF,Z.TXT,C.DDIF FOO.DDIF
```

3 In the final example, the first input file (A.DDIF) is a DDIF file, so the output file (FOO.DDIF) includes the DDIF tag as well as the contents of A.DDIF. FOO.DDIF also includes the contents of B.DDIF and C.DDIF. Again, however, the output file has no practical use because, as a result of the way DDIF files are structured, only the data from the first input file (A.DDIF) is accessible in the output file.

```
$ COPY A.DDIF,B.DDIF,C.DDIF FOO.DDIF
```

9.1.3.2 VMS Mail Utility

The VMS Mail Utility preserves the DDIF file tag when DDIF files are mailed between Version 5.1 systems. The VMS Mail Utility also preserves the DDIF file tag when you create an output file on a VMS Version 5.1 system using the EXTRACT command.

When you read a mail message that is a DDIF file, the VMS Mail Utility outputs only the text portion of the file. Similarly, if you edit a DDIF mail file, you can access only the file text; the output file is a text file that can no longer be used as a DDIF file. However, if you forward a message that consists of a DDIF file, the VMS Mail Utility sends the entire DDIF file, including DDIF semantics and the DDIF tag, to the addressee.

9.1.4 APPEND Command

This section describes what happens when you attempt to use the APPEND command in conjunction with DDIF and text files.

In the first example, the APPEND command appends a DDIF file to a text file:

```
$ APPEND X.DDIF Y.TXT
```

The output file, Y.TXT, contains its original text records as well as text from the input file, X.DDIF, reformatted as variable-length text records.

In the next example, the APPEND command appends a DDIF file to another DDIF file:

```
$ APPEND X.DDIF Y.DDIF
```

The output file, Y.DDIF, contains the DDIF tag, the original contents of Y.DDIF, and the contents of X.DDIF. However, the portion of the file that contains X.DDIF is not accessible because of the way DDIF files are structured.

In the final example, the APPEND command attempts to append a text file to a DDIF file:

```
$ APPEND X.TXT Y.DDIF
```

This append operation fails because there is no text-to-DDIF RMS extension.

9.2 DDIF Support in a Heterogeneous Environment

This section describes the implementation of DDIF support in two heterogeneous environments. The first heterogeneous environment includes VMS Version 5.1 systems and non-VMS systems. The second heterogeneous environment includes VMS Version 5.1 or earlier systems.

9.2.1 EXCHANGE/NETWORK Command

A new DCL command, EXCHANGE/NETWORK, has been created to support the transfer of files between VMS systems and non-VMS systems that do not support VMS file types. The EXCHANGE/NETWORK command transfers files in either record mode or block mode but can only be used when both systems support DECnet file transfers.

To interactively tag a DDIF file and transfer the file between a non-VMS operating system and a VMS Version 5.1 system, do the following:

1 Create the following file, assigning it the name DDIF.FDL:

```
FILE
        ORGANIZATION            sequential
        STORED_SEMANTICS        DDIF

RECORD
        CARRIAGE_CONTROL        none
        FORMAT                  fixed
        SIZE                    512
```

2 Use the following DCL command to transfer the desired file:

```
$ EXCHANGE/NETWORK/FDL=DDIF.FDL input_filespec output_filespec
```

See Chapter 10 for more information about the EXCHANGE/NETWORK command.

9.2.2 Using the COPY Command in a Heterogeneous Environment

If you use the COPY command to copy tagged DDIF files to systems other than VMS Version 5.1 systems from a VMS Version 5.1 system, the results will vary depending on the target system:

- If the target system is a non-VMS system, the file is copied, but the DDIF tag is not preserved.
- If the target system is a VMS Version 5.1 or earlier system, the copy operation fails with the VMS RMS error message RMS$_SUPPORT, network operation not supported, and a secondary error message of RMS$_SEMANTICS, inconsistent usage of RMS Semantics. Error messages similiar to the following will appear:

```
%COPY-E-OPENOUT, error opening PWEDGE::[]TRY.DDIF;1 as output
-RMS-F-SUPPORT, network operation not supported
-RMS-E-SEMANTICS, inconsistent usage of RMS Semantics
%COPY-W-NOTCOPIED, ABCD4:[DAVIDS]TRY.DDIF;1 not copied
```

- If the target system is a cluster alias for a mixed version cluster containing Version 5.1 or earlier systems, the result of the copy operation depends on whether the cluster node which actually handles the request is a Version 5.1 or earlier system.
- If you use the COPY command to copy tagged DDIF files from Version 5.1 systems to earlier systems while on an earlier system, the copy operation will fail with error message RMS$_NET, network operation failed at remote node and with a DAP status code of 16F, inconsistent usage of RMS Semantics. Error messages similiar to the following will appear:

```
%COPY-E-OPENIN, error opening ARC"davids password"::ABCD4:[DAVIDS]TRY.DDIF;1 as input
-RMS-F-NET, network operation failed at remote node; DAP code = 01F7516F
%COPY-W-NOTCOPIED, ARC"davids password"::ABCD4:[DAVIDS]TRY.DDIF;1 not copied
PWEDGE$
```

9.2.3 VMS Mail Utility in a Heterogeneous Environment

If you try to send mail messages containing DDIF files to non-VMS systems that do not support tagged files, the VMS Mail Utility returns the NOACCEPTMSG error message, indicating that the remote node cannot accept the message format.

Similarly, the VMS Mail Utility does not support the mailing of DDIF files to systems earlier than Version 5.1. As with non-VMS systems, the VMS Mail Utility returns the NOACCEPTMSG error message for systems earlier than Version 5.1, indicating that the remote node cannot accept the message format.

9.2.4 DDIF File Access Within a Mixed Version Cluster

In a cluster which contains both Version 5.1 or earlier systems, operations on DDIF files from systems earlier than Version 5.1 will cause inconsistent behavior. Records read from DDIF files on systems earlier than Version 5.1 will be fixed length 512 byte records, which contain DDIF control information in addition to the text context. Thus, typing a DDIF file on a system earlier than Version 5.1 does not produce readable text.

Copying a DDIF file using a system earlier than Version 5.1 will not preserve the DDIF tag on the output file, which will cause problems in later access to the new file from a Version 5.1 system.

However, using the Backup Utility from systems earlier than Version 5.1 will create a correct backup of DDIF files, and will properly restore DDIF files from BACKUP save-sets.

9.3 VMS RMS Interface Changes

This section provides details about the changes made to the VMS RMS interface that support access to text in VMS DECwindows DDIF files. It includes information related to tagging files and accessing tagged files through the VMS RMS interface. The section also describes how tags are preserved at the VMS RMS interface.

9.3.1 Programming Interface for File Tagging

This release note focuses on the use of the DDIF tag for supporting VMS DECwindows files, although VMS RMS also supports file tagging for other compound document data formats.

You can tag a file from the VMS RMS interface by using the $CREATE service in conjunction with a new extended attribute block (XAB) called the item XAB ($XABITM). The $XABITM macro is a general-purpose macro that was added to the RMS interface to support several Version 5.0 features. Tagged file support involves the use of the two item codes shown in Table 9–1.

Table 9–1 Tag Support Item Codes

Item	Buffer Size	Function
XAB$_STORED_SEMANTICS	64 bytes maximum	Defines the file semantics established when the file is created
XAB$_ACCESS_SEMANTICS	64 bytes maximum	Defines the file semantics desired by the accessing program

The entries XAB$_STORED_SEMANTICS and XAB$_ACCESS_SEMANTICS in the item list can represent either a control (set) function or a monitor (sense) function that can be passed to VMS RMS from the application program by way of the RMS interface.

The symbolic value XAB$K_SEMANTICS_MAX_LEN represents the tag length. This value may be used to allocate buffer space for sensing and setting stored semantics for the DDIF file.

Within any one $XABITM, you can activate either the set function or the sense function for the XAB$_STORED_SEMANTICS and XAB$_ACCESS_SEMANTICS items, because a common field (XAB$B_MODE) determines which function is active. If you want to activate both the set function and the sense function for either or both items, you must use two $XABITM control blocks, one for setting the functions and one for sensing the functions.

Each entry in the item list addressed by the $XABITM is made up of three longwords and a longword 0 that terminates the list. You can locate the item list anywhere within the readable address space for a process, but any buffers required by the related function must be located in read/write memory. If the item list is invalid, RMS returns a status of RMS$_XAB in the RAB$L_STS field and the address of the XAB in RAB$L_STV.

The format and arguments of the $XABITM macro are as follows. Note that the block length field and the type code field are statically initialized by the $XABITM macro, or may be explicitly initialized using a high-level language.

FORMAT

$XABITM *ITEMLIST=item-list-address,*
MODE= { *sensemode* / *setmode* },
NXT=next-xab-address

Arguments

The ITEMLIST argument defaults to 0 but a valid pointer must be specified when you use a XABITM. MODE defaults to *sensemode*. The symbolic offset, size, and a brief description of each XABITM field are described in the following list:

- The block length field (XAB$B_BLN) is a 1-byte static field that defines the length of the XABITM, in bytes. This field is initialized to the value XAB$C_ITMLEN.
- The type code (XAB$B_COD) field is a 1-byte static field that identifies this control block as a XABITM. This field is initialized to the value XAB$C_ITM.
- The XAB$L_ITEMLIST field is a longword field that contains the symbolic address of the item list.
- The XAB$B_MODE field is a 1-byte field that specifies whether the items can be set by the program. It contains either the symbolic value XAB$K_SETMODE or the symbolic value XAB$K_SENSEMODE (default).
- The XAB$L_NXT field is a longword field that contains the symbolic address of the next XAB in the XAB chain. A value of 0 (the default) indicates that the current XAB is the last (or only) XAB in the chain.

Example 9–1 illustrates a BLISS–32 program that tags a file through the RMS interface. The tag value shown is a 6-byte hexadecimal number representing the code for the DDIF tag. The VMS RMS program interface accepts only hexadecimal tag values.

To write to a tagged file without using an RMS extension, the application program must specify access semantics that match the file's stored semantics. As shown in the example, the $CREATE service tags the file and the $CONNECT service specifies the appropriate access semantics.

Example 9–1 Tagging a File

```
MODULE TYPE$MAIN (
        IDENT = 'X-1',
        MAIN = MAIN,
        ADDRESSING_MODE (EXTERNAL=GENERAL)
        ) =
BEGIN
!
FORWARD ROUTINE
    MAIN : NOVALUE;                          ! Main routine
!
! INCLUDE FILES:
!
LIBRARY 'SYS$LIBRARY:LIB';
OWN
    NAM             : $NAM(),
    RETLEN,
    DDIF_TAG         : BLOCK[ 7, BYTE]
                INITIAL( BYTE(%X'2B',%X'0C',%X'87',%X'73',%X'01',%X'03',%X'01')),
    FAB_XABITM        :
                $xabitm
                  ( itemlist=
                        $ITMLST_UPLIT
                          (
                            (ITMCOD=XAB$_STORED_SEMANTICS,
                             BUFADR=DDIF_TAG,
                             BUFSIZ=%ALLOCATION(DDIF_TAG))
                          ),
                    mode = SETMODE),
    RAB_XABITM        :
                $xabitm
                  ( itemlist=
                        $ITMLST_UPLIT
                          (
                            (ITMCOD=XAB$_ACCESS_SEMANTICS,
                             BUFADR=DDIF_TAG,
                             BUFSIZ=%ALLOCATION(DDIF_TAG))
                          ),
                    mode = SETMODE),
    FAB             : $FAB( fnm = 'TAGGED-FILE.TEST',
                            nam = NAM,
                            mrs = 512,
                            rfm = FIX,
                            fac = <GET,PUT,UPD>,
                            xab = FAB_XABITM),
    REC               : BLOCK[512,BYTE],
    STATUS,
    RAB             : $RAB( xab = RAB_XABITM,
```

(continued on next page)

Example 9–1 (Cont.) Tagging a File

```
                        fab = FAB,
                        rsz = 512,
                        rbf = REC,
                        usz = 512,
                        ubf = REC),
    DESC            : BLOCK[8,BYTE] INITIAL(0);
ROUTINE MAIN : NOVALUE =
BEGIN
STATUS = $CREATE( FAB = FAB );
IF NOT .STATUS
THEN
    SIGNAL (.STATUS);
STATUS = $CONNECT( RAB = RAB );
IF NOT .STATUS
THEN
    SIGNAL (.STATUS);
STATUS = $CLOSE( FAB = FAB );
IF NOT .STATUS
THEN
    SIGNAL (.STATUS);
END;
END
ELUDOM
```

9.3.2 Accessing a Tagged File

This section provides details of how VMS RMS handles access to tagged files at the program level. When a program accesses a tagged file, VMS RMS must determine whether and when to associate an RMS extension with the access. This is important to the programmer because an RMS extension may change the attributes of the accessed file.

For example, a DDIF file is stored as a sequentially organized file having 512-byte, fixed-length records. If the DDIF-to-text RMS extension is used to extract text from a DDIF file, the accessed file appears as a sequentially organized file having variable-length records with a maximum record size of 2048 bytes and an implicit carriage return.

One consideration in determining whether an access requires the RMS extension is the type of access (FAB$B_FAC). When an application program opens a file through the VMS RMS program interface, it must specify if it will be doing record I/O (default), block I/O (BIO), or mixed I/O (BRO), where the program has the option of using either block I/O or record I/O for each access. For example, if block I/O operations are specified, VMS RMS does not associate the RMS extension with the file access.

Another consideration is whether the program senses the tag when it opens a file. If the program does not sense the tag when it opens a DDIF file for record access, VMS RMS associates the RMS extension during the $OPEN and returns the file attributes that have been modified by the extension.

The final consideration is the access semantics the program specifies and the file's stored semantics (tag). If the program specifies block I/O (FAB$V_BIO) operations, RMS does not associate the RMS extension and the $OPEN service returns the file's stored attributes to the accessing program regardless of whether the program senses tags.

9.3.2.1 File Accesses That Do Not Sense Tags

This section describes what happens when a program does not use the XABITM to sense a tag when it opens a file.

When a program opens a DDIF file for record operations and does not sense the tag, VMS RMS assumes that the program wants to access text in the file. In this case, VMS RMS associates the RMS extension, which provides file attributes that correspond to record-mode access.

When a program opens a DDIF file with the FAB$V_BRO option and does not sense the tag, any subsequent attempt to use block I/O fails. If the program specifies block I/O (FAB$V_BIO) when it invokes the $CONNECT service, the operation fails because the file attributes returned at $OPEN permit record access only. Similarly, if the program specifies the FAB$V_BRO option when it opens the file, and then specifies mixed mode (block/record) operations by not specifying RAB$V_BIO at $CONNECT time, block operations such as READ and WRITE are disallowed.

9.3.2.2 File Accesses That Sense Tags

VMS RMS does not associate the RMS extension as part of the $OPEN service if a program opens a DDIF file and senses the stored semantics. This allows the program to specify access semantics with the $CONNECT service. VMS RMS returns the file attributes, including the stored semantics attribute (tag value), to the program as part of the $OPEN service.

When the program subsequently invokes the $CONNECT service, VMS RMS uses the specified operations mode to determine its response. If the program specified FAB$V_BRO with the $OPEN service and then specifies block I/O (RAB$V_BIO) when it invokes the $CONNECT service, VMS RMS does not associate the RMS extension.

But if the program specifies record access or FAB$V_BRO when it opens the file and then decides to use record I/O when it invokes the $CONNECT service, VMS RMS compares the access semantics with the file's stored semantics to determine whether to associate the RMS extension. If the access semantics match the stored semantics, VMS RMS does not associate the RMS extension. If the access semantics do not match the stored semantics, VMS RMS associates the access with the RMS extension. In this case, the program must use the $DISPLAY service to obtain the modified file attributes. If VMS RMS cannot find the appropriate RMS extension, the operation fails and the $CONNECT service returns the EXTNOTFOU error message.

If the application program senses the file's stored semantics, VMS RMS allows mixed-mode operations. In this case, mixed block and record operations are permitted because the application gets record mode file attributes and data from the RMS extension and block mode file attributes and data from the file.

Example 9–2 illustrates a BLISS–32 program that accesses a tagged file from an application program that does not use an RMS extension.

Example 9–2 Accessing a Tagged File

```
MODULE TYPE$MAIN (
        IDENT = 'X-1',
        MAIN = MAIN,
        ADDRESSING_MODE (EXTERNAL=GENERAL)
        ) =
BEGIN
!
FORWARD ROUTINE
    MAIN : NOVALUE;                          ! Main routine
!
! INCLUDE FILES:
!
LIBRARY 'SYS$LIBRARY:STARLET';
OWN
    NAM             : $NAM(),
    ITEM_BUFF    : BLOCK[ XAB$K_SEMANTICS_MAX_LEN,BYTE ],
    RETLEN,
    FAB_XABITM         :
                $xabitm
                  ( itemlist=
                        $ITMLST_UPLIT
                          ((ITMCOD=XAB$_STORED_SEMANTICS,
                             BUFADR=ITEM_BUFF,
                             BUFSIZ=XAB$K_SEMANTICS_MAX_LEN,
                             RETLEN=RETLEN)),
                    mode = SENSEMODE),
    RAB_ITEMLIST  : BLOCK[ ITM$S_ITEM + 4, BYTE ],
    RAB_XABITM    : $XABITM
                  ( itemlist=RAB_ITEMLIST,
                    mode=SETMODE ),
    FAB           : $FAB( fnm = 'TAGGED-FILE.TEST',
                          nam = NAM,
                          fac = <GET,PUT,UPD>,
                          xab = FAB_XABITM),
    REC             : BLOCK[512,BYTE],
    STATUS,
    RAB           : $RAB( xab = RAB_XABITM,
                          fab = FAB,
                          rsz = 512,
                          rbf = REC,
                          usz = 512,
                          ubf = REC),
    DESC            : BLOCK[8,BYTE] INITIAL(0);
ROUTINE MAIN : NOVALUE =
BEGIN
STATUS = $OPEN( FAB = FAB );
IF NOT .STATUS
THEN
    SIGNAL (.STATUS);
RAB_ITEMLIST[ ITM$W_BUFSIZ ] = .RETLEN;
RAB_ITEMLIST[ ITM$L_BUFADR ] = ITEM_BUFF;
RAB_ITEMLIST[ ITM$W_ITMCOD ] = XAB$_ACCESS_SEMANTICS;
STATUS = $CONNECT( RAB = RAB );
IF NOT .STATUS
THEN
```

(continued on next page)

Example 9–2 (Cont.) Accessing a Tagged File

```
    SIGNAL (.STATUS);
STATUS = $CLOSE( FAB = FAB );
IF NOT .STATUS
THEN
    SIGNAL (.STATUS);
END;
END
ELUDOM
```

9.3.3 Preserving Tags

In order to preserve the integrity of a tagged file that is being copied or transmitted, the tag must be preserved in the destination (output) file. The most efficient way to use the RMS interface for propagating tags is to open the source file (input) and sense the tag using a $XABITM with the item code XAB$_STORED_SEMANTICS:

```
    .
    .
    .
ITEMLIST[ ITM$W_BUFSIZ ] = XAB$K_SEMANTICS_MAX_LEN;
ITEMLIST[ ITM$L_BUFADR ] = ITEM_BUFF;
ITEMLIST[ ITM$L_RETLEN ] = RETLEN;
ITEMLIST[ ITM$W_ITMCOD ] = XAB$_STORED_SEMANTICS;
    .
    .
    .
XABITM[ XAB$B_MODE ] = XAB$K_SENSEMODE;
STATUS = $OPEN( FAB = FAB );
    .
    .
    .
```

Then create the destination (output) file and set the tag using a $XABITM with the item code XAB$_STORED_SEMANTICS:

```
    .
    .
    .
IF .RETLEN GTR 0
THEN
    BEGIN
    ITEMLIST[ ITM$W_ITMCOD ] = XAB$_STORED_SEMANTICS;
    ITEMLIST[ ITM$L_SIZE   ] = .RETLEN;
    XABITM[ XAB$B_MODE ] = XAB$K_SETMODE;
    END;

STATUS = $CREATE( FAB = FAB );
    .
    .
    .
END;
END
ELUDOM
```

9.4 Distributed File System Support for DDIF Tagged Files

Version 1.1 of the Distributed File System (DFS) includes limited support for DDIF tagged files. You can create and read DDIF files on a DFS device when the DFS client node is running VMS Version 5.1. You can also use the DIRECTORY/FULL command to determine whether a DDIF file on a DFS device is tagged.

You cannot use the SET FILE/[NO]SEMANTICS command either to tag DDIF files or to remove the tags from DDIF files on a DFS device. Furthermore, the Backup Utility does not preserve the DDIF tag or the DDIF stored semantics for data files on a DFS device.

9.5 VMS RMS Errors

Four VMS RMS error messages signal the user when the appropriate error condition exists:

- RMS$_EXTNOTFOU
- RMS$_SEMANTICS
- RMS$_EXT_ERR
- RMS$_OPNOTSUP

The RMS$_EXTNOTFOU error message indicates that VMS RMS has not found the specified RMS extension. Verify that the file is correctly tagged, using the DIRECTORY/FULL command, and that the application program is specifying the appropriate access semantics.

VMS RMS returns the RMS$_SEMANTICS error message when you try to create a tagged file on a remote system earlier than VMS Version 5.1 from a Version 5.1 system.

VMS RMS returns the RMS$_EXT_ERR error when the DDIF RMS extension detects an inconsistency.

VMS RMS returns the RMS$_OPNOTSUP error when the RMS DDIF extension is invoked by an RMS operation. For example, if the extension does not support write access to a DDIF file, verify that the application program is not performing record operations that modify the file.

10 EXCHANGE/NETWORK Command

This chapter describes the DCL command EXCHANGE/NETWORK and its error messages.

EXCHANGE/NETWORK

The EXCHANGE/NETWORK command allows the VMS operating system to transfer files to or from operating systems that do not support VMS file organizations. The transfer occurs over a DECnet network communications link that connects VMS and non-VMS operating system nodes.

Using DECnet services, the EXCHANGE/NETWORK command can:

- Transfer files between a VMS node and a non-VMS system node
- Transfer a group of input files to a group of output files
- Transfer files between two non-VMS nodes, provided those nodes share DECnet connections with the VMS node that issues the EXCHANGE/NETWORK command

The EXCHANGE/NETWORK command imposes the following restrictions:

- Transfers of files can occur only between disk devices. (If a disk device is not the desired permanent residence for the file, you must either move the file to a disk before issuing the command or retrieve the file from a disk after the command completes.)
- The remote system must have a block size of 512 bytes, where a byte is 8 bits long.
- The nodes transferring files must support the DECnet Data Access Protocol (DAP).

The VMS Record Management Services (RMS) facility provides VMS access to records in VMS RMS files. To transfer VMS RMS files between two nodes where both nodes are VMS nodes, use one of the other DCL commands (such as COPY, APPEND, or CONVERT), as appropriate. These commands recognize RMS file organizations and are designed to ensure that RMS record structures are preserved as your files are moved.

Use the EXCHANGE/NETWORK command to transfer files between VMS nodes and non-VMS nodes when the differences in the file organizations would otherwise prevent the transfer or could lead to undesirable results. While COPY ensures that both the contents and the attributes of a replicated file are preserved, EXCHANGE/NETWORK is more flexible. EXCHANGE/NETWORK offers you explicit control of your record attributes during file transfers, with the opportunity to make a file usable on several different operating systems.

FORMAT

EXCHANGE/NETWORK *input-file-spec[, . . .]*
output-file-spec

PARAMETERS

input-file-spec[, . . .]

Specifies the name of an existing file to be transferred. Wildcard characters are allowed. Use a comma (,) to indicate multiple file specifications.

output-file-spec

Specifies the name of the output file into which the input is transferred.

You must specify at least one field in the output file specification. If you omit the device or directory, your current default device and directory are used. The EXCHANGE/NETWORK command replaces any other missing fields (file name, file type, version number) with the corresponding field of the input file specification.

EXCHANGE/NETWORK creates a new output file for every input file that you specify.

You can use the asterisk wildcard character in place of the following: file name, file type, or version number. The EXCHANGE/NETWORK command uses the corresponding field in the related input file to name the output file. You can also use the wildcard character in the output file specification to direct EXCHANGE/NETWORK to create more than one output file. For example:

```
$ EXCHANGE/NETWORK A.A,B.B  MYPC::*.C
```

This EXCHANGE/NETWORK command creates the files A.C and B.C at the non-VMS target node MYPC.

A more complete explanation of wildcard characters and version numbers follows in the Description section.

DESCRIPTION

The EXCHANGE/NETWORK command transfers files between VMS nodes and non-VMS nodes connected to the same DECnet network. If the non-VMS system does not support VMS file organizations, EXCHANGE/NETWORK can modify or discard file and record attributes during the transfer. However, if the target system is a VMS node, you have the option of applying new file and record attributes to the output file by supplying a File Definition Language (FDL) file, as described later in this section. EXCHANGE/NETWORK provides a number of defaults to handle the majority of transfers properly. However, in some situations, you need to know your file or record format requirements at both nodes.

VMS File and Record Attributes

All RMS files in the VMS environment include stored information, known as the file and record attributes, to describe the file and record characteristics. File attributes consist of items such as file organization, file protection, and file allocation information. Record attributes consist of items such as the record format, record size, key definitions for indexed

files, and carriage control information. These attributes define the data format and access methods for the VMS RMS facility.

Non-VMS operating systems that do not support VMS file organizations have no means of storing file and record attributes with their files. Transferring a VMS file to a non-VMS system that is unable to store and handle file and record attributes can result in most of this information being discarded. Removing these attributes from a file can render it useless if it must be returned to the VMS system.

Transferring Files to VMS Nodes

When you transfer files to a VMS system from a non-VMS system, the files typically assume default file and record attributes. However, you can specify the attributes that you want the file to acquire in a File Definition Language (FDL) file. If you specify an FDL file with the /FDL qualifier, the FDL file determines the characteristics of the output file. This feature is useful in establishing compatible file and record attributes when you transfer a file from a non-VMS system to a VMS system. However, when you use an FDL file, you also assume responsibility for determining the required characteristics.

See the *VMS File Definition Language Facility Manual* for more information on FDL files.

Transferring Files to Non-VMS Nodes

EXCHANGE/NETWORK discards file and record attributes associated with a VMS file during a transfer to a non-VMS system that does not support VMS file organizations. Be aware that the loss of file and record attributes in the transfer can render the output file useless for many applications.

Selecting Transfer Modes

The EXCHANGE/NETWORK command has four transfer mode options: AUTOMATIC, BLOCK, RECORD, and CONVERT. For most file transfers, AUTOMATIC is sufficient. The AUTOMATIC transfer mode option allows EXCHANGE/NETWORK to transfer files using either block or record I/O. The selection is based on the input file organization and the operating systems involved.

Selecting the BLOCK transfer mode option forces EXCHANGE/NETWORK to open both the input and output files for block I/O access. The input file is then transferred to the output file block by block. Use this transfer mode when you transfer executable images. It is also useful when you must preserve a file's content exactly, which is a common requirement when you store files temporarily on another system or when cooperating applications exist on the systems.

Selecting the RECORD transfer mode option forces EXCHANGE/NETWORK to open both the input file and output file for record I/O access. The input file is then transferred to the output file record by record. This transfer mode is primarily used for transferring text files.

Selecting the CONVERT transfer mode option forces EXCHANGE/NETWORK to open the input file for RECORD access and the output file for BLOCK access. Records are then read in from the input file, packed into blocks, and written to the output file. This transfer mode is primarily used for transferring files with no implied carriage control. For example, to transfer a file created with DIGITAL Standard Runoff (DSR) to a DECNET-DOS system, you must use the CONVERT transfer mode option. To transfer the resultant output file back to a VMS node, use the AUTOMATIC transfer mode option.

Wildcard Characters

Wildcard characters are permitted in the file specifications and follow the behavior typical of other VMS commands with respect to the VMS node.

When more than one input file is specified, but wildcards are not specified in the output file specification, the first input file is copied to the output file, and each subsequent input file is transferred and given a higher version number of the same output file name. Note that the files are not concatenated into a single output file. Also note that when you transfer files to foreign systems that do not support version numbers, only one output file results, and it is the last input file.

To create multiple output files, specify multiple input files and use at least one of the following:

- An asterisk wildcard character in the output file name, file type, or version number field
- Only a node name, a device name, or a directory specification as the output file specification

When you create multiple output files, EXCHANGE/NETWORK uses the corresponding field from each input file in the output file name.

Use the /LOG qualifier when you specify multiple input and output files to verify that the files were copied as you intended.

Version Numbers

The following guidelines apply when the target node file formats accept version numbers.

If no version numbers are specified for input and output files, the EXCHANGE/NETWORK command (by default) assigns a version number to the output files that is either of the following:

- The version number of the input file
- A version number one greater than the highest version number of an existing file with the same file name and file type

When the output file version number is specified by an asterisk wildcard character, the EXCHANGE/NETWORK command uses the version numbers of the associated input files as the version numbers of the output files.

If the output file specification has an explicit version number, the EXCHANGE/NETWORK command normally uses that number for the output file specification. However, if an equal or higher version of the output file already exists, no warning message is issued, the file is copied, and the version number is set to a value one greater than the highest version number already existing.

File Protection and Creation/Revision Dates

The EXCHANGE/NETWORK command treats an output file as a new file when any portion of the output file name is explicitly specified. When the output node is a VMS system, the creation date for a new file is set to the current time and date. However, if the output file specification consists *only* of wildcard characters, the output file no longer qualifies as a new file, and, therefore, the creation date of the input file is used. That is, if the output file specification is one of the following, the creation date becomes that of the input file: *, *.*, or *.*;*.

The revision date of the output file is always set to the current time and date; the backup date is set to zero. The output file is assigned a new expiration date. (Expiration dates are set by the file system if retention is enabled; otherwise, they are set to zero.)

When the target node is a VMS node, the protection and access control list (ACL) of the output file is determined by the following parameters, in the following order:

1 Protection of previously existing versions of the output file

2 Default protection and ACL of the output directory

3 Process default file protection

For an introduction to access control lists, see the *VMS DCL Concepts Manual*.

On VMS systems, the owner of the output file usually is the same as the creator of the output file. However, if a user with extended privileges creates the output file, the owner is either the owner of the parent directory or the owner of a previous version of the output file, if one exists.

Extended privileges include any of the following:

- SYSPRV or BYPASS
- System UIC
- GRPPRV if the owner of the parent directory (or previous version of the output file) is in the same group as the creator of the new output file
- An identifier (with the resource attribute) representing the owner of the parent directory (or previous version of the output file)

QUALIFIERS

/BACKUP

Modifies the time value specified with the /BEFORE or /SINCE qualifier. /BACKUP selects files according to the dates of their most recent backup. This time qualifier is incompatible with the other time qualifiers that also allow you to select files according to time attributes: /CREATED, /EXPIRED, and /MODIFIED. If you do not specify any of these four time qualifiers, the default is /CREATED.

/BEFORE[=time]

Selects only those files dated prior to the specified time. You can specify time as an absolute time, as a combination of absolute and delta times, or as one of the following keywords: TODAY (default), TOMORROW, or YESTERDAY. Specify one of the following time qualifiers with /BEFORE to indicate the time attribute to be used as the basis for selection: /BACKUP, /CREATED (default), /EXPIRED, or /MODIFIED.

See the *VMS DCL Concepts Manual* for complete information about specifying time values.

/BY_OWNER[=uic]

Selects only those files whose owner user identification code (UIC) matches the specified owner UIC. The default UIC is that of the current process.

Specify the UIC using standard UIC format as described in the *VMS DCL Concepts Manual*.

/CONFIRM
/NOCONFIRM (default)

Controls whether a request is issued before each file transfer operation to confirm that the operation should be performed on that file. The following responses are valid:

YES	NO	QUIT
TRUE	FALSE	CTRL/Z
1	0	ALL
	RET	

You can use any combination of uppercase and lowercase letters for word responses. Word responses can be abbreviated to one or more letters (for example, T, TR, or TRU for TRUE), but these abbreviations must be unique. Affirmative answers are YES, TRUE, and 1. Negative answers are NO, FALSE, 0, and the RETURN key. QUIT or CTRL/Z indicates that you want to stop processing the command at that point. When you respond with ALL, the command continues to process, but no further prompts are given. If you type a response other than one of those in the list, DCL issues an error message and redisplays the prompt.

/CREATED (default)

Modifies the time value specified with the /BEFORE or /SINCE qualifier. The /CREATED qualifier selects files based on their date of creation. This time qualifier is incompatible with the other time qualifiers that also allow you to select files according to time attributes: /BACKUP, /EXPIRED, and /MODIFIED. If you do not specify any of these four time qualifiers, the default is /CREATED.

/EXCLUDE=(file-spec[, . . .])

Excludes the specified files from the file transfer operation. You can include a directory but not a device in the file specification. Wildcard characters are allowed in the file specification. However, you cannot use relative version numbers to exclude a specific version. If you provide only one file specification, you can omit the parentheses.

/EXPIRED

Modifies the time value specified with the /BEFORE or /SINCE qualifiers. /EXPIRED selects files according to their expiration date. (The expiration date is set with the SET FILE/EXPIRATION_DATE command.) This time qualifier is incompatible with the other time qualifiers that also allow you to select files according to time attributes: /BACKUP, /CREATED, and /MODIFIED. If you do not specify any of these four time qualifiers, the default is /CREATED.

/FDL=fdl-file-spec

Specifies that the output file characteristics are described in the File Definition Language (FDL) file. Use this qualifier when you require special output file characteristics. See the *VMS File Definition Language Facility Manual* for more information about FDL files.

Use of the /FDL qualifier implies that the transfer mode is block by block. However, the transfer mode you specify with the /TRANSFER_MODE qualifier prevails.

/LOG
/NOLOG (default)

Controls whether the EXCHANGE/NETWORK command displays the file specifications of each file copied.

When you use the /LOG qualifier, the EXCHANGE/NETWORK command displays the following for each copy operation: (1) the file specifications of the input and output files, and (2) the number of blocks or the number of records copied (depending on whether the file is copied on a block-by-block or record-by-record basis).

/MODIFIED

Modifies the time value specified with the /BEFORE or /SINCE qualifier. The /MODIFIED qualifier selects files according to the date on which they were last modified. This time qualifier is incompatible with the other time qualifiers that also allow you to select files according to time attributes: /BACKUP, /CREATED, and /EXPIRED. If you do not specify any of these four time qualifiers, the default is /CREATED.

/SINCE[=time]

Selects only those files dated after the specified time. You can specify time as an absolute time, a combination of absolute and delta times, or as one of the following keywords: TODAY (default), TOMORROW, or YESTERDAY. Specify one of the following time qualifiers with /SINCE to indicate the time attribute to be used as the basis for selection: /BACKUP, /CREATED (default), /EXPIRED, or /MODIFIED.

See the *VMS DCL Concepts Manual* for complete information about specifying time values.

/TRANSFER_MODE=option

Specifies the I/O method to be used in the transfer. This qualifier is useful for all file formats. You can specify any one of the following options:

Option	Function
AUTOMATIC	Allows EXCHANGE/NETWORK to determine the appropriate transfer mode.
BLOCK	Transfers block by block.
CONVERT[=option[, . . .]]	Reads records from the input file, packs them into blocks, and writes to the output file in block mode. The options determine what additional information is inserted during the transfer.
RECORD	Transfers record by record.

The AUTOMATIC transfer mode option allows EXCHANGE/NETWORK to determine the appropriate transfer mode. The default is the AUTOMATIC transfer mode.

If you explicitly select the BLOCK transfer mode option, EXCHANGE/NETWORK opens both the input and output files for block I/O. EXCHANGE/NETWORK then transfers the files block by block.

If you explicitly select the RECORD transfer mode option, EXCHANGE/NETWORK opens both the input and output files for record I/O. The target system must support record operations, and the input file must be record oriented.

If you select the CONVERT transfer mode option, EXCHANGE/NETWORK reads records in from the input file, packs them into blocks, and writes them to the output file in block mode. There are four options available with the CONVERT transfer mode to control the insertion of special characters in the records, as explained in the following paragraphs:

- CARRIAGE_CONTROL
- COUNTED
- FIXED_CONTROL
- RECORD_SEPARATOR=separator

If you specify CARRIAGE_CONTROL, any carriage control information in the input file is interpreted, expanded into actual characters, and included with each record.

If you specify COUNTED, the length of each record in bytes is included at the beginning of each record. The length includes all FIXED_CONTROL, CARRIAGE_CONTROL, and RECORD_SEPARATOR information in each record.

If you specify FIXED_CONTROL, all variable length with fixed control record (VFC) information is written to the output file as part of the data. This information follows the record length information if the COUNTED option was specified.

If you specify RECORD_SEPARATOR, a 1- or 2-byte record separator is inserted between each record. Record separator characters are the last characters in the record. The three choices for separator characters are CR for carriage return only, LF for line feed only, or CRLF for carriage return and line feed.

EXAMPLES

1
```
$ EXCHANGE/NETWORK VMS_FILE.DAT FOO::FOREIGN_SYS.DAT
```

In this example, the EXCHANGE/NETWORK command transfers the file VMS_FILE.DAT located in the current default device and directory to the file FOREIGN_SYS.DAT on the non-VMS node FOO. Because the /TRANSFER_MODE qualifier was not explicitly specified, EXCHANGE/NETWORK automatically determines whether the transfer method should be block or record I/O.

2
```
$ EXCHANGE/NETWORK/TRANSFER_MODE=BLOCK -
_$ FOO::FOREIGN_SYS.DAT VMS_FILE.DAT
```

In this example, the EXCHANGE/NETWORK command transfers the file FOREIGN_SYS.DAT from the non-VMS node FOO to the file VMS_FILE.DAT in the current default device and directory. Block I/O is specified for the transfer mode.

3
```
$ EXCHANGE/NETWORK/FDL=VMS_FILE_DEFINITION.FDL -
_$ FOO::REMOTE_FILE.TXT  VMS_FILE.DAT
```

In this example, the EXCHANGE/NETWORK command transfers the file REMOTE_FILE.TXT on node FOO to the file VMS_FILE.DAT. The file attributes for the output file VMS_FILE.DAT are obtained from the File Definition Language (FDL) source file VMS_FILE_DEFINITION.FDL. For more information about creating FDL files, see the *VMS File Definition Language Facility Manual*. Because the qualifier /FDL is specified and the /TRANSFER_MODE qualifier is omitted, the transfer mode uses block I/O, by default.

4
```
$ EXCHANGE/NETWORK -
_$ /TRANSFER_MODE=CONVERT=(CARRIAGE_CONTROL,COUNTED, -
_$ RECORD_SEPARATOR=CRLF,FIXED_CONTROL) -
_$ PRINT_FILE.TXT  FOO::*
```

In this example, the EXCHANGE/NETWORK command transfers the file PRINT_FILE.TXT from the current default device and directory to the file PRINT_FILE.TXT on the non-VMS node FOO. The use of the CONVERT option with the /TRANSFER_MODE qualifier forces the input file to be read in record by record, modified as specified by the convert options described below, and written to the output file block by block. As many records as will fit are packed into the output blocks.

The CONVERT option CARRIAGE_CONTROL specifies that carriage control information is converted to ASCII characters and inserted before the data or appended to the record, depending on whether prefix control or postfix control, or both, are used. The CONVERT option FIXED_CONTROL specifies that any fixed control information be translated to ASCII characters and inserted at the beginning of the record. The

CONVERT option RECORD_SEPARATOR=CRLF appends the two specified characters, carriage return and line feed, to the end of the record. The CONVERT option COUNTED specifies that the total length of the record must be counted (once the impact of all the previous convert options have been added), and the result is to be inserted at the beginning of the record, in the first two bytes.

10.1 EXCHANGE/NETWORK Error Messages

This section provides an alphabetical summary of the messages that can be generated by the DCL command EXCHANGE/NETWORK. A number of the messages are common to other utilities and DCL commands such as the Exchange Utility and the DCL command COPY.

For information about the VMS style of presenting error messages, see the *VMS System Messages and Recovery Procedures Reference Manual: Part I.*

It may be helpful to remove this section and file it as a temporary addendum to the *VMS System Messages and Recovery Procedures Reference Volume* until the next manual update becomes available. As an alternative, you might want to manually add just the following messages, which are the ones unique to the EXCHANGE/NETWORK command: BADDEV, CLOSEIN, CLOSEOUT, FDLPARSE, and RECPRN.

10.1.1 Messages

ATPC, at PC = 'xxxxxxxx'

Facility: Shared by several facilities

Explanation: This message generally accompanies a message indicating a software failure.

User Action: Take corrective action based on the accompanying messages.

BADDEV, device is unsupported for transfer

Facility: Exchange/Network Command

Explanation: You attempted to transfer the file either to or from a device that the command does not support, such as magnetic tape.

User Action: Correct an error in the device specification, if that is causing the problem, or try to use a different device.

BADLOGIC, internal logic error detected, please report error 'errorcode'

Facility: Shared by several facilities

Explanation: The Exchange Utility or EXCHANGE/NETWORK command encountered an unexpected condition and terminated.

User Action: Please submit a Software Performance Report (SPR), describing the error number and the commands that resulted in the message. If the error is only reproducible using a particular piece of media, send a copy of the media with the SPR. (Use the BACKUP/PHYSICAL command to make the copy.)

CLOSEIN, error closing 'filespec' as input

Facility: Exchange/Network Command

Explanation: The EXCHANGE/NETWORK command encountered an error while closing an input file. This message is usually accompanied by a VMS RMS message indicating the reason for the failure.

User Action: Take corrective action based on the associated message.

CLOSEOUT, error closing 'filespec' as output

Facility: Exchange/Network Command

Explanation: The EXCHANGE/NETWORK command encountered an error while closing the output file. This message is usually accompanied by a VMS RMS message indicating the reason for the failure.

User Action: Take corrective action based on the accompanying message.

COPIEDB, 'input-filespec' copied to 'output-filespec' ('nnn' blocks)

Facility: Shared by several facilities

Explanation: The message displays the number of blocks copied, based on the size of the input files. This message is informational.

User Action: None.

COPIEDC, 'input-filespec' copied to 'output-filespec' ('nnn' records packed into 'mmm' blocks)

Facility: Shared by several facilities

Explanation: The message displays the number of records read in from the input file and the number of blocks written to the output file. This message is informational.

User Action: None.

COPIEDR, 'input-filespec' copied to 'output-filespec' ('nnn' records)

Facility: Shared by several facilities

Explanation: The message displays the number of records copied, based on the size of the input files. This message is informational.

User Action: None.

FDLPARSE, fatal error encountered parsing FDL file

Facility: Exchange/Network Command

Explanation: An error occurred during the parsing of the FDL file supplied by the user. This message is issued with an accompanying message.

User Action: Take corrective action based on the accompanying message and reenter the command.

HIGHVER, higher version of 'output-filespec' already exists

Facility: Shared by several facilities

Explanation: An explicit version number is requested for an output file; the directory already contains an entry for the same file name and file type with a higher version number. This message is informational. Note that if the file is subsequently specified in a command, the system will locate the previously existing version if no version number is specified. The newly created output file will not be used.

User Action: None.

OPENIN, error opening 'input-filespec' as input

Facility: Shared by several facilities

Explanation: An input file cannot be opened. This message is usually accompanied by a VMS RMS message indicating the reason for the failure.

User Action: Take corrective action based on the associated message.

OPENOUT, error opening 'output-filespec' as output

Facility: Shared by several facilities

Explanation: An output file cannot be opened. This message is usually accompanied by a VMS RMS message indicating the reason for the failure.

User Action: Take corrective action based on the associated message.

READERR, error reading 'filespec'

Facility: Shared by several facilities

Explanation: An input file specified cannot be read. This message is usually accompanied by a VMS RMS message indicating the reason for the failure.

User Action: Take corrective action based on the associated message.

RECPRN, 'filespec' contained 'nnn' records with invalid PRN fields ('mmm' prefix 'ppp' postfix)

Facility: Exchange/Network Command

Explanation: The specified file contained one or more records with invalid printing control fields in the print file control area (PRN). The first byte of the control area constitutes a prefix area while the second byte constitutes a postfix area. These areas specify the carriage control to be performed before and after printing, respectively. A carriage return and a line-feed character were substituted for each invalid byte, and the file was copied.

User Action: If the substitution is not acceptable, determine the cause of the PRN field error, fix it, and reenter the command.

SYNTAX, error parsing 'string'

Facility: Shared by several facilities

Explanation: The command syntax is invalid. The message displays the rejected portion of the command.

User Action: Use the DCL command HELP or refer to the *VMS DCL Dictionary* for the correct syntax and reenter the command.

WRITEERR, error writing 'filespec'

Facility: Shared by several facilities

Explanation: The output file cannot be written. This message should be accompanied by a VMS RMS message indicating the reason for the error.

User Action: Follow the recovery procedure for the specified VMS RMS message.

11 SET/SHOW DISPLAY Commands

This chapter describes the DCL commands SET DISPLAY and SHOW DISPLAY. Please note that these commands can only be used if you have DECwindows installed.

SET DISPLAY

Allows the creation and modification of workstation display devices. The workstation display device is used as a mechanism to store the necessary information about how to access the workstation.

FORMAT

SET DISPLAY *[display device]*

PARAMETERS

display device

If you specify the /CREATE qualifier, the parameter **display device** is defined as a logical name to point to this new device. If no device string is specified, DECW$DISPLAY is used. When a DECwindows program is run, it uses the logical name DECW$DISPLAY to find the workstation.

If the /CREATE qualifier is not specified, then the device is modified based on the values of the other qualifiers specified.

QUALIFIERS

/CREATE

Creates the display device by cloning the WSA0 device.

/[NO]PERMANENT

Defaults to /PERMANENT if /CREATE is also used, which makes a device permanent even if the reference count of assigned channels goes to zero. Specifying a SET DISPLAY WSAn/NOPERMANENT causes the device to be deleted when the last channel is deassigned.

/NODE=workstation

Defines the node to be associated with this display device. When /CREATE is specified, NODE defaults to SYS$REM_NODE if defined, otherwise to SYS$NODE. This behavior allows the SET DISPLAY/CREATE command to be used whether you are logged in to the workstation locally, or are logged into a remote system after having entered the SET HOST command from the workstation.

/TRANSPORT=transport-name

Defines the transport to be used. Defaults to DECnet if /CREATE is also specified. Local is another acceptable value, provided the server and client are on the same node.

/SERVER=server-number

Defines the server to be associated with this display device. Defaults to 0 if /CREATE is also specified.

/SCREEN=screen-number

Defines the screen to be associated with this display device. Defaults to 0 if /CREATE is also specified.

SHOW DISPLAY

Displays the characteristics of a display device.

FORMAT

SHOW DISPLAY *[display device]*

PARAMETERS

display device

The parameter **display device** is used as the device name to list. If no device is specified, the logical name DECW$DISPLAY is used.

Index

A

B

C

D

E

F

G

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local DIGITAL subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local DIGITAL subsidiary or approved distributor
Internal[1]	————	SDC Order Processing - WMO/E15 *or* Software Distribution Center Digital Equipment Corporation Westminster, Massachusetts 01473

[1]For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

VMS Version 5.1
New Features Manual
AA–MG29A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

--- Do Not Tear - Fold Here and Tape ---

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

--- Do Not Tear - Fold Here ---

Cut Along Dotted Line

Reader's Comments

VMS Version 5.1
New Features Manual
AA–MG29A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

-- Do Not Tear - Fold Here and Tape ----------

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

-- Do Not Tear - Fold Here ----------

Cut Along Dotted Line